FSA Modernization Partner

United States Department of Education

Federal Student Aid

EAI Release 3.0

Build and Test Report

Task Order #80

Deliverable #80.1.3

September 6, 2002

Table of Contents

1 EXECUTIVE SUMMARY

Enterprise Application Integration (EAI) is a key component of FSA's Modernization Program. It is a technology that supports integration of data, systems, applications, and business processes across the FSA enterprise. EAI does this by creating a central communication infrastructure called the "EAI Bus". The EAI Bus consists of a scalable, extensible architecture comprised of two clustered servers, which form a "hub", thereby supporting architecture across the systems using the EAI Bus.

In order for FSA's systems and applications to use the services provided by EAI, the EAI team works with the client team to design, build, test, and deploy interfaces. These interfaces enable transportation, translation, formatting, and routing of messages.

This report describes:

- The interface architecture
- The test procedures for interfaces migrated to the production environment during Release 3.0 of EAI.

For mapping between the proposed work for Release 3.0 and this deliverable, please refer to Appendix A.

1.1 Objectives

This report documents the build and test procedures and results for EAI Release 3.0 interfaces in order to provide FSA managers, application teams and other stakeholders with a reference guide.

The tests outlined in this report are based on the functional scenarios developed to validate the previously designed MQSeries Messaging and Transformation activities.

1.2 Approach

These steps were followed to build and test interfaces:

1. EAI works with application teams to gather, build, and test requirements.
2. Create an Interface Control Document (ICD) from the requirements. This describes how to build an interface and describes testing.
3. Develop a test plan.
4. Execute the test plan.
5. Record test results.

1.3 Description of Sections

This deliverable is divided into the following sections:

- Section 1 – Executive Summary

  The Executive Summary provides an introduction and overview of the EAI Build and Test Report.

- Section 2 – EAI Component Build

  This section contains design and build documentation for interfaces deployed to the production environment in EAI Release 3.0.

- Section 3 – EAI Test Methodology

  The EAI Test Methodology focuses on the validation of the architectural design of the Release 3.0 EAI Core Architecture. The test scenario descriptions will provide the objective and an overview of the test to be performed, function(s) exercised, and any other pertinent aspects of the test scenario. Test scenario inputs, expected results, and acceptance criteria are discussed.

- Section 4 – EAI Component Tests

  The EAI Component Tests document the detailed component test plan for each legacy system. Diagrams are used throughout this section to explain the flow of data between the different EAI components.

- Section 5 – Source Code for New Adapters

  This section addresses the executive summary document statement that the Build and Test report “Provides the source code for new adapters”.

- Appendices A-L

  Appendices are included in this deliverable to provide detailed documentation. These appendices are referenced throughout the document.

  Appendix A sets the context for this deliverable with regard to all EAI Release 3.0 deliverables. Appendices B-K include assembly test conditions. Appendix L includes a timeline for EAI Release 3.0.

1.4 Scope

This document captures test results from Assembly Tests performed by the EAI team. The test results in this deliverable are for interfaces built and tested during EAI Release 3.0. EAI shares testing responsibilities with the application teams listed below. Application teams are responsible for the following tests:

- User Acceptance Testing (UAT)
- Inter-Systems Testing (IST)
- Schools Testing
- Operational Readiness Testing (ORT)
- Performance Testing

MQSeries messaging and transformation activities were developed for the following Release 3.0 FSA systems:

- Common Origination and Disbursement (COD)
- Central Processing System (CPS)
- Direct Loan Servicing System (DLSS)
- Direct Loan Origination System (DLOS)
- Electronic Campus Based System (eCBS)
- Financial Accounting and Reporting System (FARS)
- Financial Partner Data Mart (FP Data Mart)
- Financial Management System (FMS)
- LO Web System (LO Web)
- National Student Loan Data System (NSLDS)
- Post-Secondary Education Participants System (PEPS)
- Student Aid Internet Gateway (SAIG/bTrade)

The Build portion of this document ensures that all required components defined in Release 3.0 of the EAI Core Architecture are installed, configured, and operational.

The Test portion ensures that the actual outputs produced conform to the expected outputs as defined by each test scenario.

Appendix L displays the Assembly Test schedule for EAI Release 3.0.

1.5 Intended Audience

The EAI Build and Test Report is a reference document for application teams and FSA.

2 EAI COMPONENT BUILD

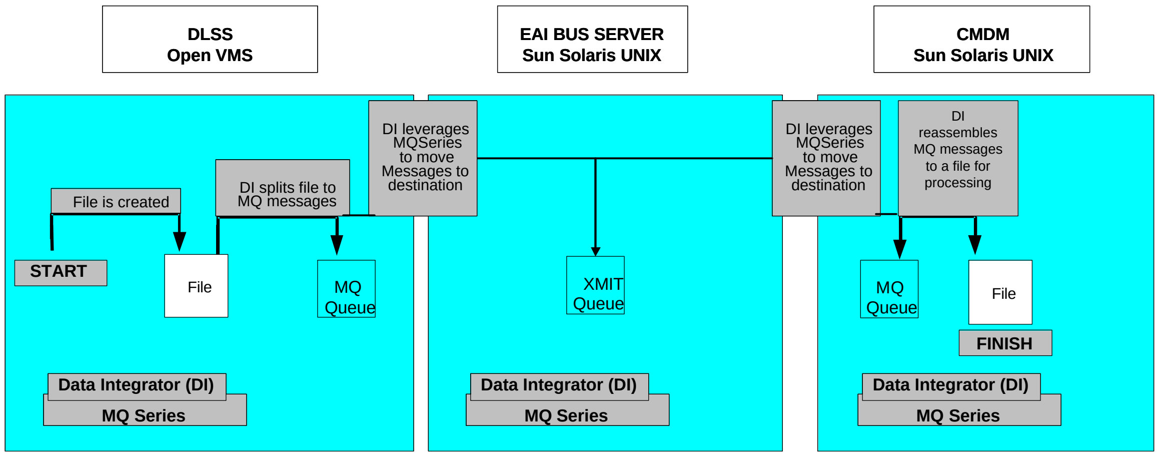
This section contains design descriptions for interfaces deployed to the production environment during EAI Release 3.0. The EAI team captured detailed design information in two documents: Interface Control Documents (ICD) and Internal Interface Documents (IID). The ICD describes interface requirements and functions as an interface agreement between the EAI team and Application team. The IID is used internally and provides detailed design information required for interface development. Due to the large volume of design documentation, the ICDs and IIDs have not been included in this deliverable, but are available upon request. The requirements information and the following diagrams have been extracted from ICD's and IIDs respectively. The following sections contain the design diagrams and descriptions summarizing the application interfaces that were designed, developed, tested, installed, configured, and deployed by the EAI team.

Adapter source code was developed during EAI Release 3.0 by the EAI team. For further information on source code, please see section 5 of this deliverable.

2.1 FARS Retirement DLSS – CMDM Interface Design Description

The sample function selected for the DLSS to CMDM interface validates the ability to send DLSS data from the send script to the CMDM system. Once the source file is completed (process is done writing to it), Data Integrator moves the flat file from DLSS to the appropriate destination location in the CMDM system via MQSeries queues and channels.

The figure below describes the message flow for the batch DLSS to CMDM interface.



The flow of a bulk file from DLSS to CMDM, via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. An Open VMS step is called to Data Integrator sender adapter to initiate transfer of the file.
3. The Sender adapter splits the file into MQ messages, and the source (local) Queue Manager (DLSST1) on DLSS moves the message to the specified XMIT queue (EAID1).

4. The message is transmitted directly to the destination Receiver (Queue Manager WAST1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (WAST1.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. The source file is deleted on a successful file transfer to CMDM.

Files

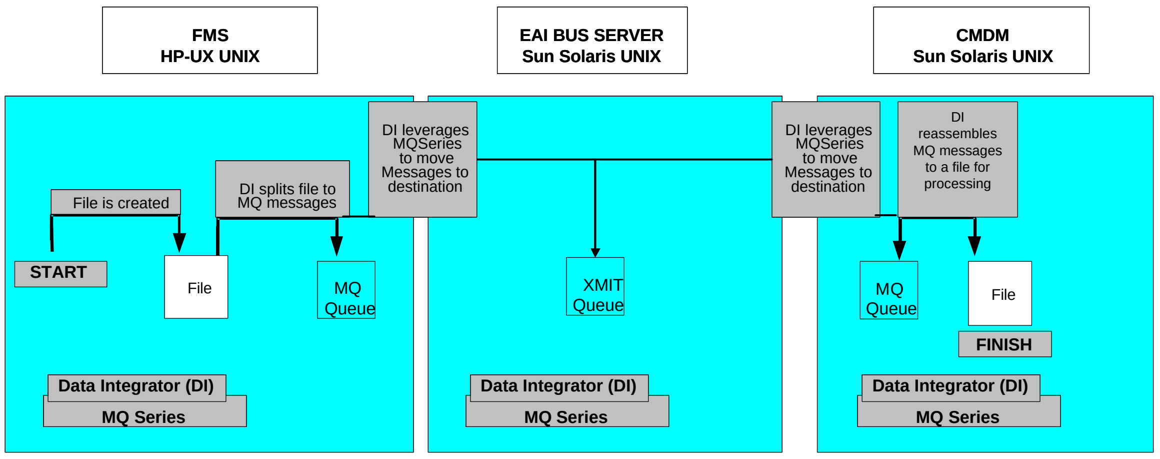
The following input files were defined on the DLSS for access by the EAI Core test application in execution of this test scenario:

Filenames	Function
MIS_LOANS.txt	MIS Loads File
BORROWERS.txt	Borrowers File
FICE_SCHOOL_CODES.txt	FICE School File

2.2 FARS Retirement FMS – CMDM Interface Design Description

The sample function selected for the FMS to CMDM interface validates the ability to send DLSS data from the send script to the CMDM system. Once the source file is completed (process is done writing to it), Data Integrator will move the flat file from FMS to the appropriate destination location in the CMDM system via MQSeries queues and channels.

The figure below describes the message flow for the batch FMS to CMDM interface.



The flow of a bulk file from FMS to CMDM, via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. A script is called to Data Integrator sender adapter to initiate transfer of the file.
3. The Sender adapter splits the file into MQ messages, and the source (local) Queue Manager (FMST1) on FMS moves the message to the specified XMIT queue (EAID1).
4. The message is transmitted directly to the destination Receiver (Queue Manager WAST1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (WAST1.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. The source file is deleted on a successful file transfer to CMDM.

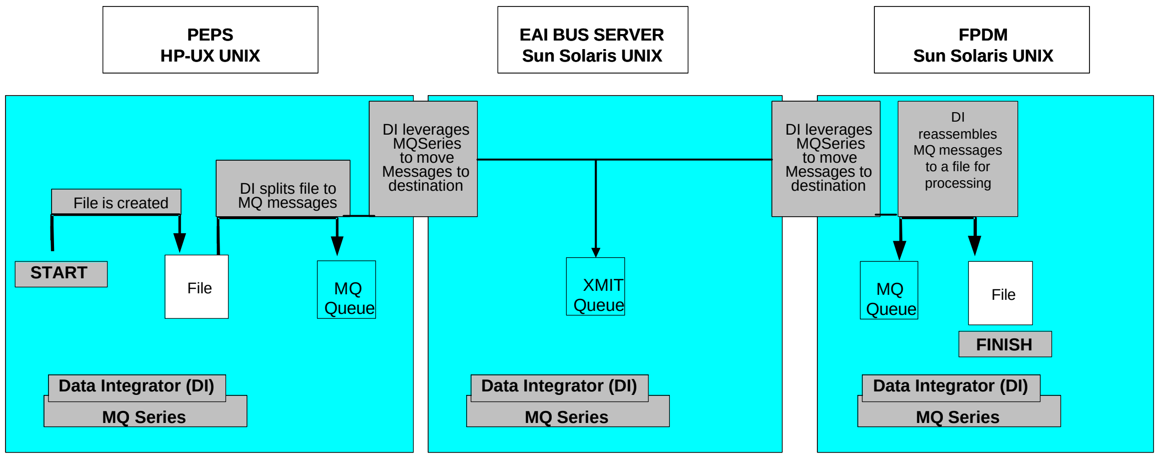
Files

The following input files were defined on the FMS for access by the EAI Core test application in execution of this test scenario:

Filenames	Function
cmdm_if010	IF10 File
cmdm_if020	IF20 File
cmdm_grec	G Record File
cmdm_manual	Manual Transaction File

2.3 Financial Partner Data Mart PEPS – FPDM Interface Design Description

The sample function selected for the PEPS to FPDM interface validates the ability to send PEPS data from the send script to the FPDM system. Once the source file is completed (process is done writing to it), Data Integrator will move the flat file from PEPS to the appropriate destination location in the FPDM system. The figure below describes the message flow for the batch PEPS to FPDM interface.



The flow of a bulk file from PEPS to FPDM, via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. A script is called to Data Integrator sender adapter to initiate transfer of the file.
3. The Sender adapter splits the file into MQ messages, and the source (local) Queue Manager (PEPSI1) on PEPS moves the message to the specified XMIT queue (EAII2).
4. The message is transmitted directly to the destination Receiver (Queue Manager FPDMI). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (FDPMI.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and then submits an operational reply to the originating Manager.
8. The source file is deleted on a successful file transfer to FPDM.

Files

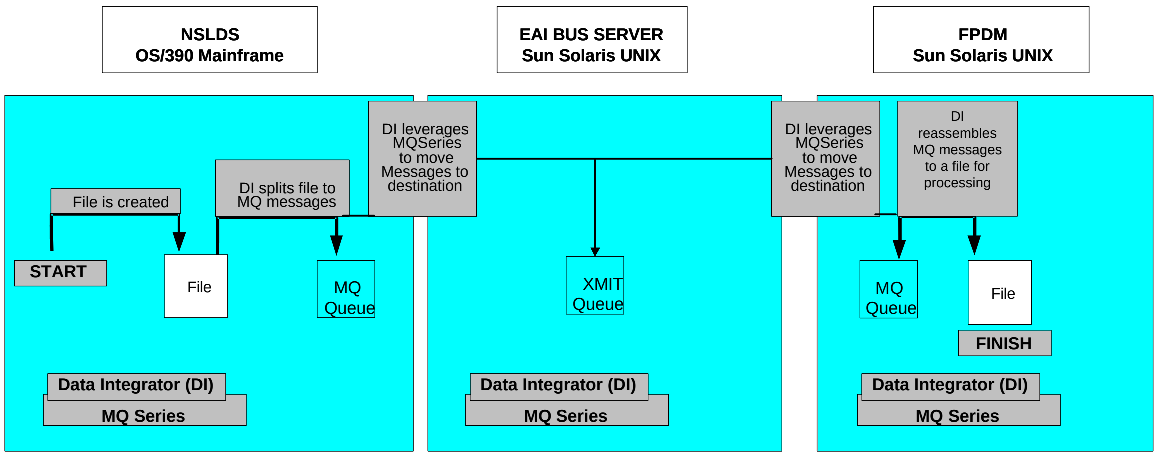
The following input files were defined on the PEPS for access by the EAI Core test application in execution of this test scenario:

Filenames	Function
f_lndr_audit.txt	PEPS Extract containing audit information by lender
f_lndr_audit_dfcncy.txt	PEPS Extract containing audit deficiency and payment information by lender
f_lndr_pgm_review.txt	PEPS Extract containing program review information by lender
f_lndr_pgm_review_dfcncy.txt	PEPS Extract containing program review deficiency and payment information by lender
f_closed_sch.txt	PEPS Extract containing information on closed schools

2.4 Financial Partner Data Mart NSLDS – FPDM Interface Design Description

The sample function selected for the NSLDS to FPDM interface validates the ability to send NSLDS data from the send script to the FPDM system. Once the source file is completed (process is done writing to it), Data Integrator will move the flat file from NSLDS to the appropriate destination location in the FPDM system via MQSeries queues and channels.

The figure below describes the message flow for the batch NSLDS to FPDM interface.



The flow of a bulk file from NSLDS to FPDM, via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. A script is called to Data Integrator sender adapter to initiate transfer of the file.
3. The Sender adapter splits the file into MQ messages, and the source (local) Queue Manager (NPD1) on NSLDS moves the message to the specified XMIT queue (EAII2).
4. The message is transmitted directly to the destination Receiver (Queue Manager FPDMI). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (FDPMI.B) and appropriately transmitted via the MQSeries message channel agent.

5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. The source file is deleted on a successful file transfer to FPDM.

Files

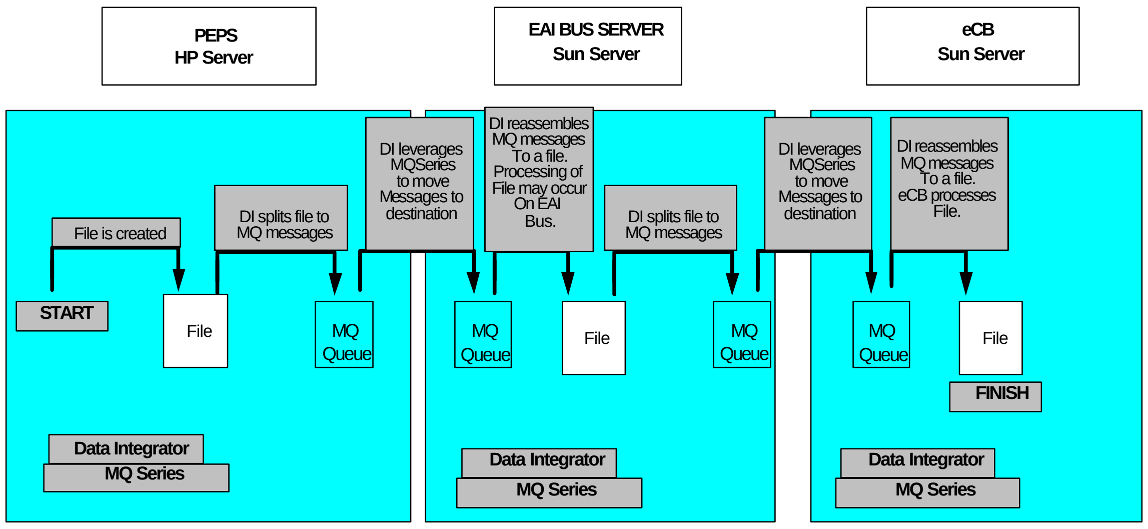
The following input files have been defined on the NSLDS for access by the EAI Core test application in execution of this test scenario:

Filenames	Function
dmffeloan.txt	NSLDS extract containing portfolio information and market share by lender/guarantee agency
dmclaim.txt	NSLDS extract containing closed school and false certification claims by guarantee agency
dmvfa.txt	NSLDS extract containing VFA performance measures by guarantee agency
dmlenderf.txt	NSLDS extract containing cohort default rate by lender
dmgasum.txt	NSLDS extract containing aggregate descriptor and dollar amount by guarantee agency

2.5 eCampus Based PEPS - eCB Design Description

The sample function selected for the PEPS to eCB interface validates the ability to send PEPS school data using the send script to the eCB system. Once the source data is completely written to the file system, the Data Integrator dirmon process initiates the move of the flat file from PEPS to the appropriate destination location on the EAI bus via MQSeries queues and channels. After the file is successfully written to the bus, a script running on the bus initiates the move of the flat file to the appropriate destination location on the eCB system.

The figure below describes the message flow for the batch PEPS to eCB interface.


PEPS
HP Server
EAI BUS SERVER
Sun Server
eCB
Sun Server
File is created
DI splits file to MQ messages
DI leverages MQSeries to move Messages to destination
DI reassembles MQ messages To a file. Processing of File may occur On EAI Bus.
DI splits file to MQ messages
DI leverages MQSeries to move Messages to destination
DI reassembles MQ messages To a file. eCB processes File.
START
File
MQ Queue
MQ Queue
File
MQ Queue
MQ Queue
File
FINISH
Data Integrator
MQ Series
Data Integrator
MQ Series
Data Integrator
MQ Series

The flow of a bulk file from PEPS to eCB via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. The dirmon process uses the Data Integrator sender adapter to initiate transfer of the file.
3. The sender adapter splits the file into MQ messages, and the source (local) Queue Manager (PEPSI1) on PEPS moves the message to the specified XMIT queue.
4. The message is transmitted to the Receiver (EAI Queue Manager EAII2). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (EAII2.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits and operational reply to the originating Manager.
8. The source file is deleted on a successful file transfer to the EAI bus.

9. Upon successful file transfer to the bus, Data Integrator calls a script to initiate transfer of the file to eCB.
10. The sender adapter splits the file into MQ messages, and the source (local) Queue Manager (EAIP2) on the EAI bus moves the message to the specified XMIT queue.
11. The message is transmitted to the Receiver (eCB Queue Manager ECBSI2). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (ECBSI2.B) and appropriately transmitted via the MQSeries message channel agent.
12. The Sender replies to the originating Manager, indicating that the file has been submitted.
13. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
14. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
15. Upon successful file transfer to eCB, Data Integrator calls a post processor script to handle the school file.

Files

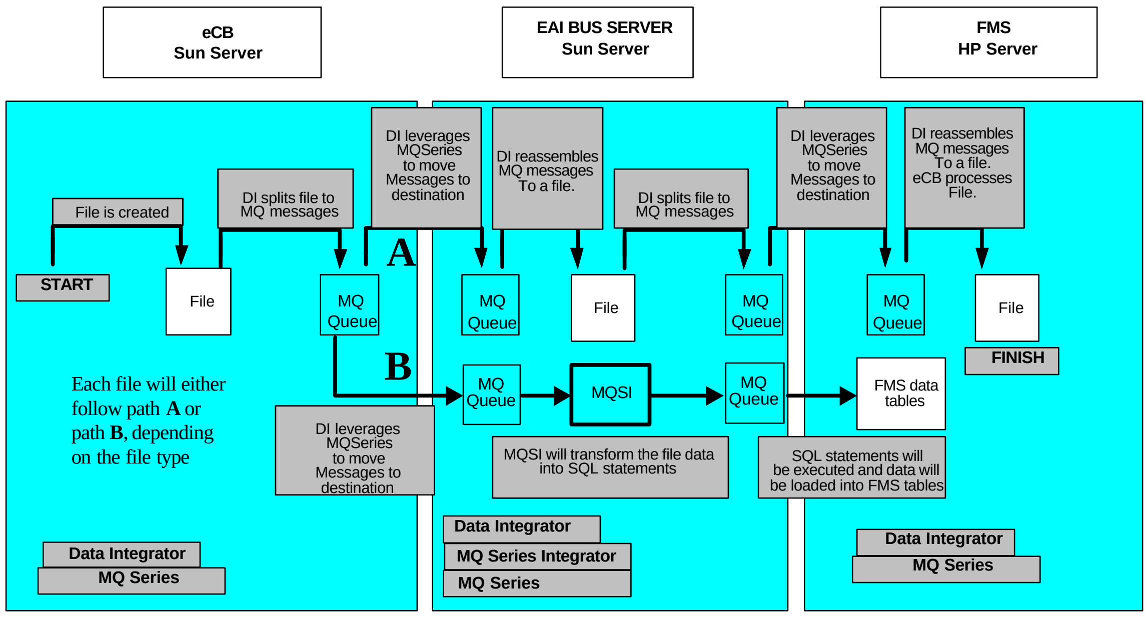
The following input files have been defined on PEPS for access by the EAI Core test application in execution of this test scenario:

Filenames	Function
schfile_extract_daily_yyyymmdd.Z	Compressed PEPS school data file

2.6 eCampus Based eCB - FMS Interface Design Description

The sample function selected for the eCB to FMS interface validates the ability to send eCB financial data using the send script to the FMS system. Once the source data is completely written to the file system, Data Integrator will initiate the move of the flat file from eCB to the appropriate destination location on the EAI bus via MQSeries queues and channels. If the file is of certain type, MQSeries Integrator (MQSI) will transform the data in the file into SQL statements and the data will then be loaded into FMS data tables. All other file types will simply pass through the EAI bus and will be directly transmitted to the destination location on the FMS system.

The figure below describes the message flow for the batch eCB to FMS interface.


eCB
Sun Server
EAI BUS SERVER
Sun Server
FMS
HP Server
File is created
START
File
DI splits file to MQ messages
MQ Queue
DI leverages MQSeries to move Messages to destination
A
B
Each file will either follow path A or path B, depending on the file type
DI leverages MQSeries to move Messages to destination
Data Integrator
MQ Series
DI reassembles MQ messages To a file.
DI splits file to MQ messages
MQ Queue
File
MQ Queue
MQ Queue
MQSI
MQ Queue
MQSI will transform the file data into SQL statements
Data Integrator
MQ Series Integrator
MQ Series
DI leverages MQSeries to move Messages to destination
DI reassembles MQ messages To a file. eCB processes File.
MQ Queue
File
FINISH
FMS data tables
SQL statements will be executed and data will be loaded into FMS tables
Data Integrator
MQ Series

The flow of a bulk file from eCB to FMS via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. A script is called to Data Integrator sender adapter to initiate transfer of the file.
3. The sender adapter splits the file into MQ messages, and the source (local) Queue Manager (ECBSI2) on eCB moves the message to the specified XMIT queue.
4. The message is transmitted to the Receiver (EAI Queue Manager EAII2). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (EAII2.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.

If the file is **not** of type ‘unpaid teacher cancellation liabilities’ (UTCL), it will traverse along path **A.**

7A.The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager. Because the file will only pass through the EAI bus, Data Integrator continues to transfer the file to the destination.

8A.The EAI bus sender adapter splits the file into MQ messages, and the source (local) Queue Manager (EAII2) on eCB moves the message to the specified XMIT queue.

9A.The message is transmitted to the Receiver (FMS Queue Manager FMSI1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (FMSI1.B) and appropriately transmitted via the MQSeries message channel agent.

10A. The Sender replies to the originating Manager, indicating that the file has been submitted.

11A. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.

12A. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.

13A. If the file is of type ‘unpaid teacher cancellation liabilities’ (UTCL), it will traverse along path **B.**

7B. Using Data Integrator, the Receiver adapter transforms the file into one MQ message and initiates an MQSI message flow. An operational reply is submitted to the originating Manager.

8B. MQSI transforms each line of the file into an SQL statement and passes the SQL statement as a message to a queue.

9B. Using a custom Java component, the SQL message is executed against the FMS database and the data is loaded into appropriate tables.

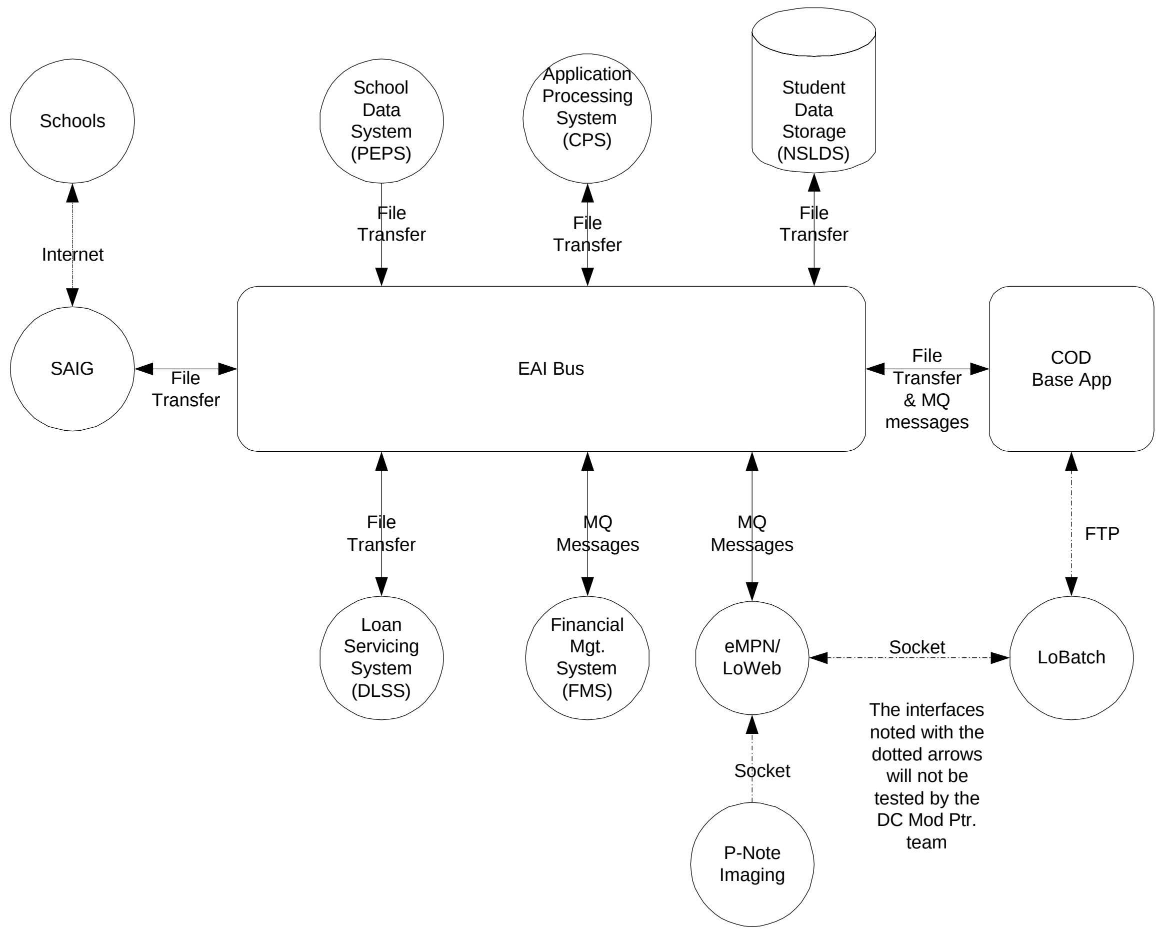
<u>Files</u>

The following input files were defined on eCB for access by the EAI Core test application in execution of this test scenario:

Filenames	Function
expmmddyyyy.txt	FISAP expenditures test file
pbsmmddyyyy.txt	Perkins Balance Sheet test file
utclmmddyyyy.txt	Unpaid Teacher Cancellation Liabilities test file
mmddyyyy.obl, mmddyyyy.obp	FSEOG/FWS/Perkins Award obligations and adjustments test file

2.7 COD Interfaces

2.7.1 COD Interfaces Summary


Schools
School Data System (PEPS)
Application Processing System (CPS)
Student Data Storage (NSLDS)
Internet
File Transfer
File Transfer
File Transfer
SAIG
File Transfer
EAI Bus
File Transfer & MQ messages
COD Base App
File Transfer
MQ Messages
MQ Messages
FTP
Loan Servicing System (DLSS)
Financial Mgt. System (FMS)
eMPN/ LoWeb
Socket
LoBatch
Socket
P-Note Imaging
The interfaces noted with the dotted arrows will not be tested by the DC Mod Ptr. team

COD interfaces via the EAI Bus to the following Trading Partners: SAIG, DLSS, NSLDS, CPS, PEPS, FMS, and eMPN. The EAI Bus transports files to/from Schools, DLSS, NSLDS, CPS, and PEPS to/from COD via MQ Series and the Data Integrator file transfer utility. The EAI Bus also transports MQ messages to/from FMS and eMPN to/from COD. The COD EAI Interfaces team will test the above interfaces between the EAI Bus, Trading Partners, and COD to ensure all files and MQ messages are delivered once and only once. The COD EAI Interfaces team tested the custom components (listed in the COD EAI Interfaces Custom Components) that manipulate the data that is delivered by the bus.

A common logging utility was used for every interface where data passed through transformation/custom Java adapters. The detailed test conditions for the common logging utility are located in Appendix K.

2.7.2 COD to CPS Interface Overview

2.7.2.1 COD to CPS Overview

All student application information is sent as a Free Application for Student Aid (FAFSA) to the Central Processing System (CPS). CPS is a mainframe application that edits the application data and performs matches against INS, SSA, VA, Selective Service, and NSLDS data to determine student eligibility. Once the data has been validated, CPS calculates the Expected Family Contribution (EFC) and generates the Institutional Student Information Report (ISIR) for the schools and state agencies, and the Student Aid Report (SAR) for the student. The ISIR and SAR contain essentially the same data, but in different formats.

COD needs a portion of the FAFSA information to process Pell and Direct Loan awards and disbursements. This information is the Abbreviated Application Record, plus its header and trailer.

CPS provides COD with data about applicants for loans and grants via the Abbreviated Applicant Record. COD provides CPS with updated School Data.

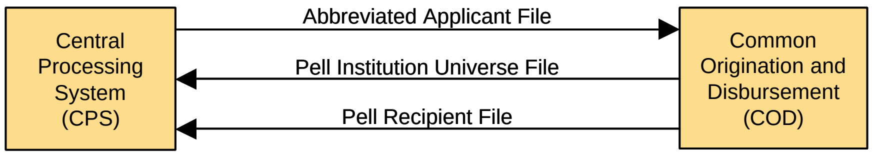
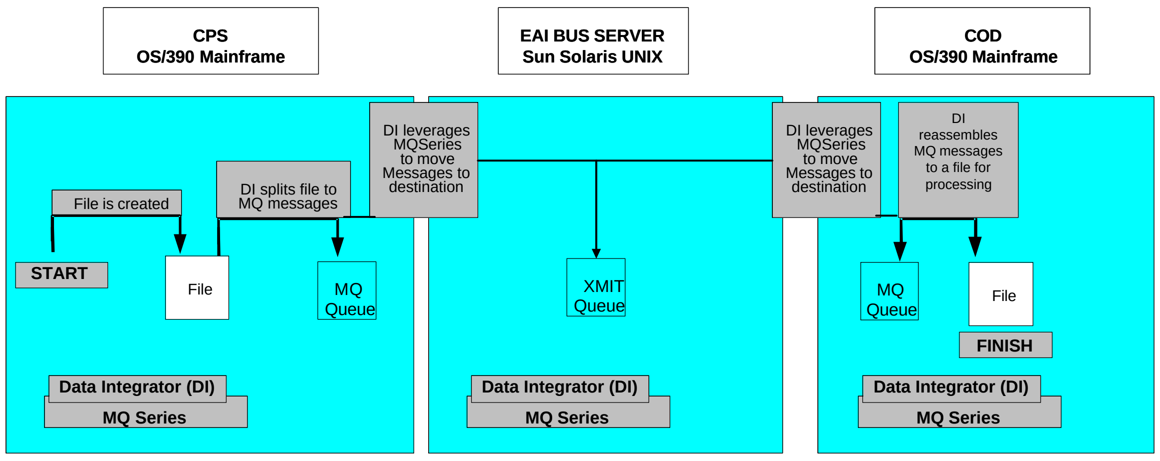

Central Processing System (CPS)
Abbreviated Applicant File
Pell Institution Universe File
Pell Recipient File
Common Origination and Disbursement (COD)

Figure 1: COD – CPS Interfaces

NOTE: EAI Bus has been abstracted out

2.7.2.2 Abbreviated Applicant File Detailed Design Description

The sample function selected for the Abbreviated Applicant File interface validates the ability to send CPS data from the send script to the COD system. Once the source file is completed (process is done writing to it), Data Integrator moves the flat file from CPS to the appropriate destination location in the COD system via MQSeries queues and channels.



The figure below describes the message flow for the Abbreviated Applicant File interface.

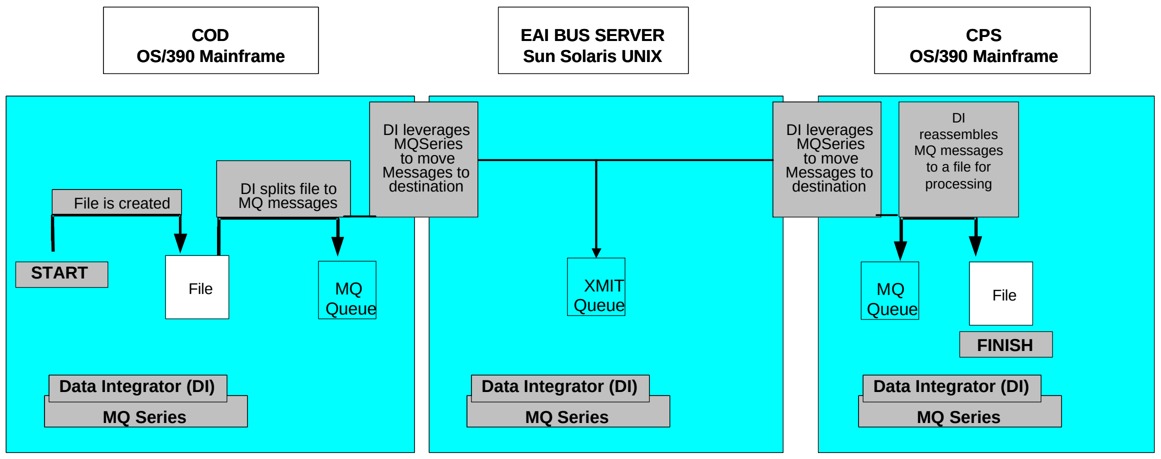
The flow of a bulk file from CPS to COD, via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. A JCL is called to Data Integrator sender adapter to initiate transfer of the file.
3. The Sender adapter splits the file into MQ messages, and the source (local) Queue Manager (QCT1) on CPS moves the message to the specified XMIT queue (EAIA1).
4. The message is transmitted directly to the destination Receiver (Queue Manager VD0Q). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (VD0Q.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits and operational reply to the originating Manager.
8. Post-Processing is kicked off upon receipt of file on COD.
9. The source file is deleted on a successful file transfer to COD.

2.7.2.3 Pell Institution Universe File Detailed Design Description

The sample function selected for the Pell Institution Universe File interface validates the ability to send COD data from the send script to the CPS system. Once the source file is completed (process is done writing to it), Data Integrator moves the flat file from COD to the appropriate destination location in the CPS system via MQSeries queues and channels.

The figure below describes the message flow for the Pell Institution Universe File interface.


COD
OS/390 Mainframe
EAI BUS SERVER
Sun Solaris UNIX
CPS
OS/390 Mainframe
File is created
START
File
DI splits file to MQ messages
MQ Queue
DI leverages MQSeries to move Messages to destination
XMIT Queue
DI leverages MQSeries to move Messages to destination
DI reassembles MQ messages to a file for processing
MQ Queue
File
FINISH
Data Integrator (DI)
MQ Series

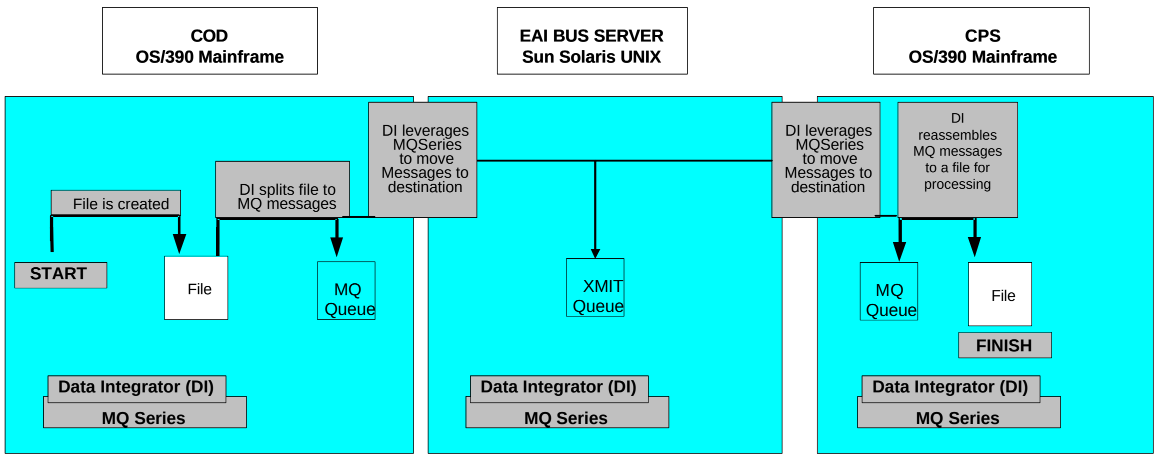
The flow of a bulk file from COD to CPS, via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. A JCL is called to Data Integrator sender adapter to initiate transfer of the file.
3. The Sender adapter splits the file into MQ messages, and the source (local) Queue Manager (VD0Q) on COD moves the message to the specified XMIT queue (EAIA1).
4. The message is transmitted directly to the destination Receiver (Queue Manager QCT1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (QCT1.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. Post-Processing is kicked off upon receipt of file on CPS.
9. The source file is deleted on a successful file transfer to CPS.

2.7.2.4 Pell Recipient File Detailed Design Description

The sample function selected for the Pell Recipient File interface validates the ability to send COD data from the send script to the CPS system. Once the source file is completed (process is done writing to it), Data Integrator moves the flat file from COD to the appropriate destination location in the CPS system via MQSeries queues and channels.

The figure below describes the message flow for the Pell Recipient File interface.



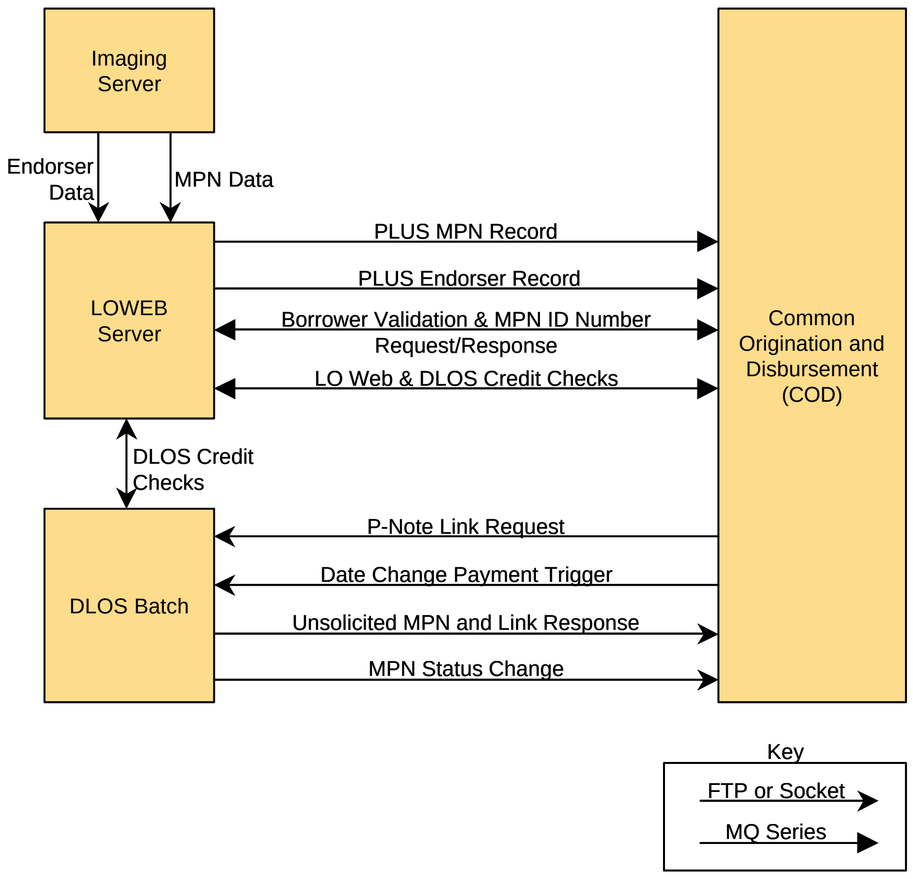
The flow of a bulk file from COD to CPS, via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. A JCL is called to Data Integrator sender adapter to initiate transfer of the file.
3. The Sender adapter splits the file into MQ messages, and the source (local) Queue Manager (VD0Q) on COD moves the message to the specified XMIT queue (EAIA1).
4. The message is transmitted directly to the destination Receiver (Queue Manager QCT1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (QCT1.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. Post-Processing is kicked off upon receipt of file on CPS.
9. The source file is deleted on a successful file transfer to CPS.

2.7.3 COD to DLOS Interface Overview

The Direct Loan Origination System (DLOS) is a single application responsible for the administration of the origination and disbursement of Direct Loans. DLOS is comprised of several components: Electronic Master Promissory Note (eMPN), Loan Origination Website (LO Web), Direct Loan Origination System (DLOS) Batch, and P-Note Imaging System (Imaging). COD requires interfaces to all of these components, some of which will be enduring and others of which will be retired.

The eMPN component provides potential borrowers with the ability to complete an application for and obtain a master promissory note (MPN) for Direct Loans via the public internet. Beginning in AY 02-03, parent borrowers will be able to apply for PLUS MPNs on the eMPN website. In AY02-03, COD will be processing all PLUS loans and will require the PLUS eMPN


Imaging Server
Endorser Data
MPN Data
LOWEB Server
PLUS MPN Record
PLUS Endorser Record
Borrower Validation & MPN ID Number Request/Response
LO Web & DLOS Credit Checks
Common Origination and Disbursement (COD)
DLOS Credit Checks
DLOS Batch
P-Note Link Request
Date Change Payment Trigger
Unsolicited MPN and Link Response
MPN Status Change
Key
FTP or Socket
MQ Series

information generated by the eMPN component and the ability to validate the existence of an PLUS loan award for potential borrowers.

The LO Web component provides access to the DLOS application for school users. School users currently have the capability to perform credit checks for potential parent borrowers from the website. COD will be performing all credit checks for AY 02-03 and an interface between the two systems will be required to continue this functionality until the LO Web component is retired.

DLOS Batch is the backend-processing component for the DLOS application and handles the linking of subsidized and unsubsidized Stafford loans for AY 02-03. COD has interfaces to request linking information and updates from the DLOS Batch component. Additionally, DLOS Batch handles prior year PLUS loans and has the ability to request credit check information from COD. This has been accomplished by passing the requests to the LO Web machine and using the same credit check interface to COD.

The P-Note Imaging component is responsible for imaging paper master promissory notes and endorser addendums. COD requires the MPN information for PLUS loans and the endorser addendum information.

Figure 2 presents the interfaces between the DLOS and COD. Additional information may be found in the following paragraphs and in the individual ICD's.

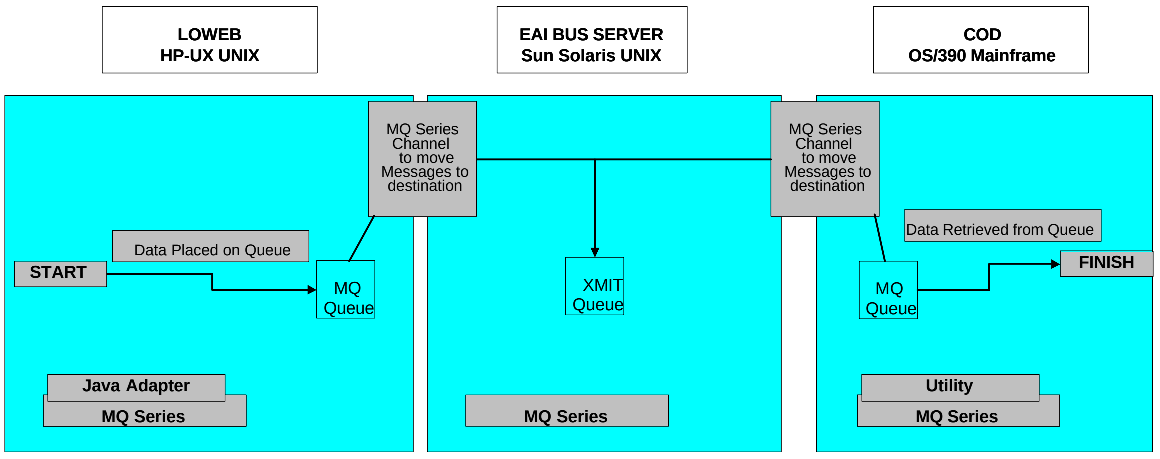
Figure 2: COD – DLOS Interfaces

NOTE: EAI Bus has been abstracted out

2.7.3.1 PLUS MPN Record Detailed Design Description

The sample function selected for the PLUS MPN Record interface validated the ability to send LOWEB data to the COD system in real time. Once the LOWEB data is placed on the queue, the channel moves the message from the LOWEB queue to the COD queue.

The figure below describes the message flow for the PLUS MPN Record interface.



The flow of a transactional real time message from LOWEB to COD, via MQSeries queue and channel is as follows:

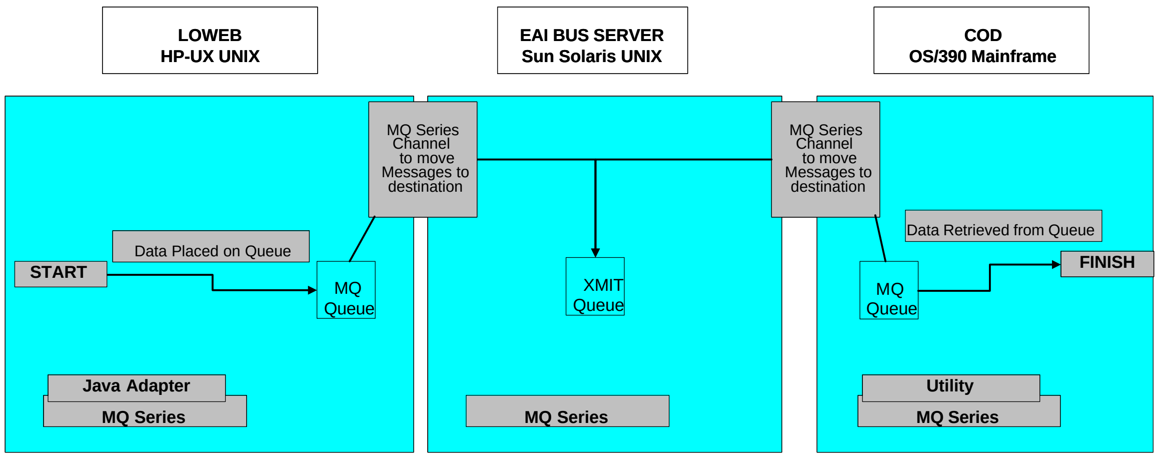
1. A PLUS MPN Record is created.
2. The socket connection service on the LOWEB Server calls the LOWEB MQSeries Java adapter to send the PLUS MPN Record.
3. The LOWEB MQSeries Java adapter places messages on the MQSeries Queue, and the source (local) Queue Manager (LOWA1) on LOWEB moves the message to the specified XMIT queue (EAIA1).

4. The message is transmitted directly to the destination Receiver (Queue Manager VD0Q). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (VD0Q.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the message has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message and submits and operational reply to the originating Manager.
8. Post-Processing is kicked off upon receipt of data on COD.

2.7.3.2 PLUS Endorser Record Detailed Design Description

The sample function selected for the PLUS Endorser Record interface validates the ability to send LOWEB data to the COD system in real time. Once the LOWEB data is placed on the queue, the channel will move the message from the LOWEB queue to the COD queue.

The figure below describes the message flow for the PLUS Endorser Record interface.



The flow of a transactional real time message from LOWEB to COD, via MQSeries queue and channel is as follows:

1. A PLUS Endorser Record is created.
2. The socket connection service on the LOWEB Server calls the LOWEB MQSeries Java adapter to send the PLUS Endorser Record.
3. The LOWEB MQSeries Java adapter places messages on the MQSeries Queue, and the source (local) Queue Manager (LOWA1) on LOWEB moves the message to the specified XMIT queue (EAIA1).
4. The message is transmitted directly to the destination Receiver (Queue Manager VD0Q). If the Receiver is on a remote queue manager (in this case), the message destination is

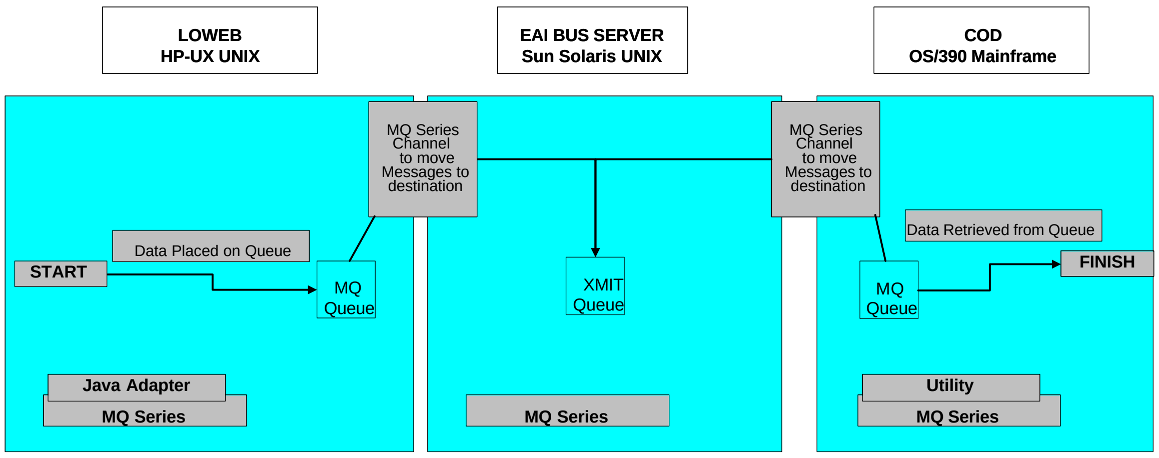
resolved to a transmission queue or set of transmission queues (VD0Q.B) and appropriately transmitted via the MQSeries message channel agent.

5. The Sender replies to the originating Manager, indicating that the message has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message and submits and operational reply to the originating Manager.
8. Post-Processing is kicked off upon receipt of data on COD.

2.7.3.3 Borrower Validation & MPN ID Number Request Detailed Design Description

The sample function selected for the Borrower Validation & MPN ID Number Request interface validates the ability to send LOWEB data to the COD system in real time. Once the LOWEB data is placed on the queue, the channel will move the message from the LOWEB queue to the COD queue.

The figure below describes the message flow for the Borrower Validation & MPN ID Number Request interface.



The flow of a transactional real time message from LOWEB to COD, via MQSeries queue and channel is as follows:

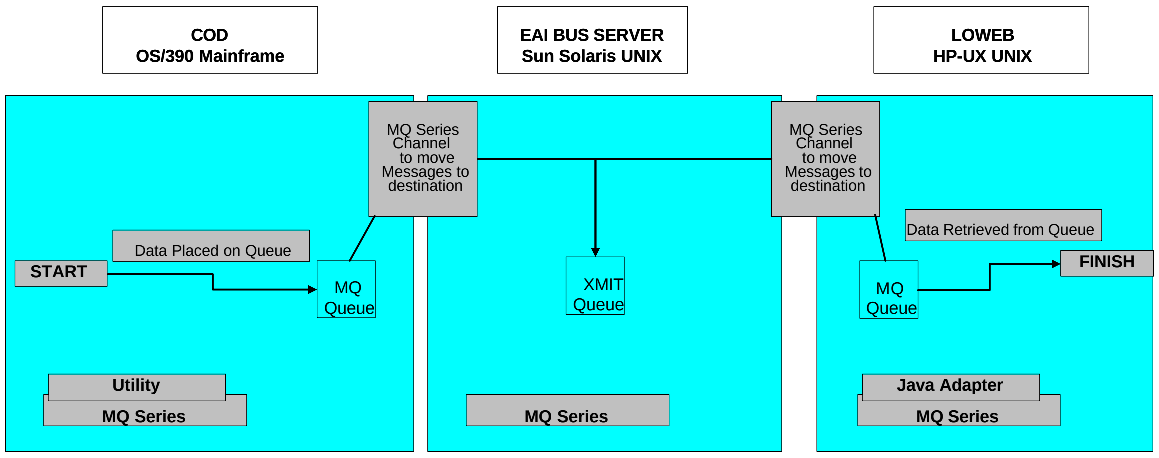
1. A Borrower Validation & MPN ID Number Request is created.
2. The socket connection service on the LOWEB Server calls the LOWEB MQSeries Java adapter to send the Borrower Validation & MPN ID Number Request.
3. The LOWEB MQSeries Java adapter places messages on the MQSeries Queue, and the source (local) Queue Manager (LOWA1) on LOWEB moves the message to the specified XMIT queue (EAIA1).

4. The message is transmitted directly to the destination Receiver (Queue Manager VD0Q). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (VD0Q.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the message has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message and submits and operational reply to the originating Manager.
8. Post-Processing is kicked off upon receipt of data on COD.

2.7.3.4 Borrower Validation & MPN ID Number Response Detailed Design Description

The sample function selected for the Borrower Validation & MPN ID Number Response interface validates the ability to send COD data to the LOWEB system in real time. Once the COD data is placed on the queue, the channel will move the message from the COD queue to the LOWEB queue.

The figure below describes the message flow for the Borrower Validation & MPN ID Number Response interface.



The flow of a transactional real time message from COD to LOWEB, via MQSeries queue and channel is as follows:

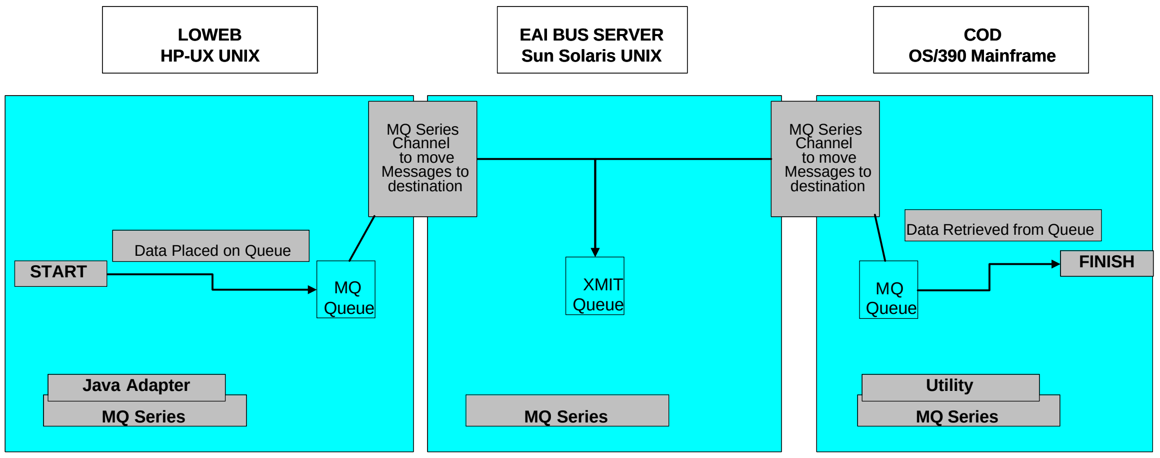
1. A Borrower Validation & MPN ID Number Response is created.
2. The Utility on the COD server places messages on the MQSeries Queue, and the source (local) Queue Manager (VD0Q) on COD moves the message to the specified XMIT queue (EAIA1).
3. The message is transmitted directly to the destination Receiver (Queue Manager LOWA1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (LOWA1.B) and appropriately transmitted via the MQSeries message channel agent.

4. The Sender replies to the originating Manager, indicating that the message has been submitted.
5. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
6. The Receiver reassembles the MQ message and submits and operational reply to the originating Manager.
7. The LOWEB MQSeries Java adapter retrieves messages from the queue.
8. Post-Processing is kicked off upon receipt of data on LOWEB.

2.7.3.5 LO Web and DLOS Credit Checks Request Detailed Design Description

The sample function selected for the LO Web and DLOS Credit Checks Request interface validates the ability to send LOWEB data to the COD system in real time. Once the LOWEB data is placed on the queue, the channel moves the message from the LOWEB queue to the COD queue.

The figure below describes the message flow for the LO Web and DLOS Credit Checks Request interface.



The flow of a transactional real time message from LOWEB to COD, via MQSeries queue and channel is as follows:

1. A LO Web and DLOS Credit Checks Request is created.
2. The socket connection service on the LOWEB Server calls the LOWEB MQSeries Java adapter to send the LO Web and DLOS Credit Checks Request.
3. The LOWEB MQSeries Java adapter places messages on the MQSeries Queue, and the source (local) Queue Manager (LOWA1) on LOWEB moves the message to the specified XMIT queue (EAIA1).
4. The message is transmitted directly to the destination Receiver (Queue Manager VD0Q). If the Receiver is on a remote queue manager (in this case), the message destination is

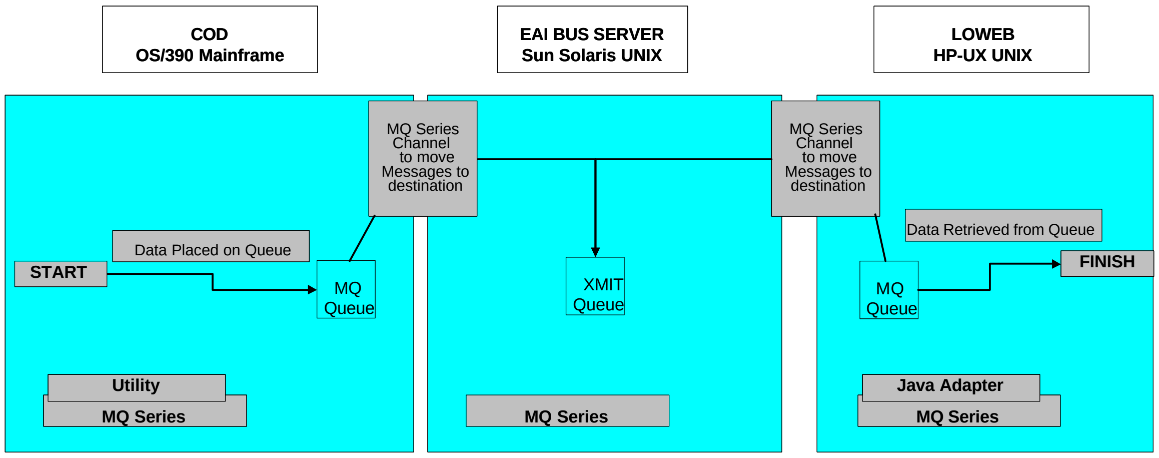
resolved to a transmission queue or set of transmission queues (VD0Q.B) and appropriately transmitted via the MQSeries message channel agent.

5. The Sender replies to the originating Manager, indicating that the message has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message and submits and operational reply to the originating Manager.
8. Post-Processing is kicked off upon receipt of data on COD.

2.7.3.6 LO Web and DLOS Credit Checks Response Detailed Design Description

The sample function selected for the LO Web and DLOS Credit Checks Response interface validates the ability to send COD data to the LOWEB system in real time. Once the COD data is placed on the queue, the channel moves the message from the COD queue to the LOWEB queue.

The figure below describes the message flow for the LO Web and DLOS Credit Checks Response interface.



The flow of a transactional real time message from COD to LOWEB, via MQSeries queue and channel is as follows:

1. A LO Web and DLOS Credit Checks Response is created.
2. The Utility on the COD server places messages on the MQSeries Queue, and the source (local) Queue Manager (VD0Q) on COD moves the message to the specified XMIT queue (EAIA1).
3. The message is transmitted directly to the destination Receiver (Queue Manager LOWA1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (LOWA1.B) and appropriately transmitted via the MQSeries message channel agent.
4. The Sender replies to the originating Manager, indicating that the message has been submitted.

5. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
6. The Receiver reassembles the MQ message and submits and operational reply to the originating Manager.
7. The LOWEB MQSeries Java adapter retrieves messages from the queue.
8. Post-Processing is kicked off upon receipt of data on LOWEB.

2.7.4 COD to DLSS Interface Overview

The Direct Loan Servicing System (DLSS) maintains loan and borrower information. DLSS tracks loans received for the life of the loan from the booking process through payment in full by the borrower or until the loan is passed to the Debt Collection Service (defaulted loan).

DLSS provides COD with loan and borrower information via the DLSS Batch Feed. Confirmations of COD-initiated transactions that took place since the last Disbursement Confirmation File was sent to COD by DLSS are contained in the Disbursement Confirmations Feed. Updates of DLSS-relevant information previously provided by COD to DLSS take place in the Daily response feed.

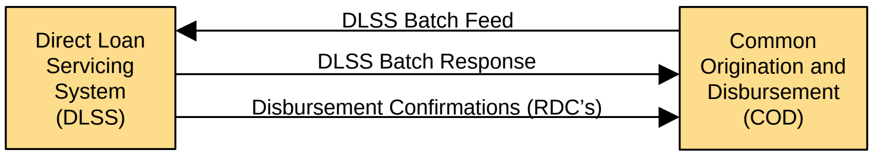

Direct Loan Servicing System (DLSS)
DLSS Batch Feed
DLSS Batch Response
Disbursement Confirmations (RDC's)
Common Origination and Disbursement (COD)

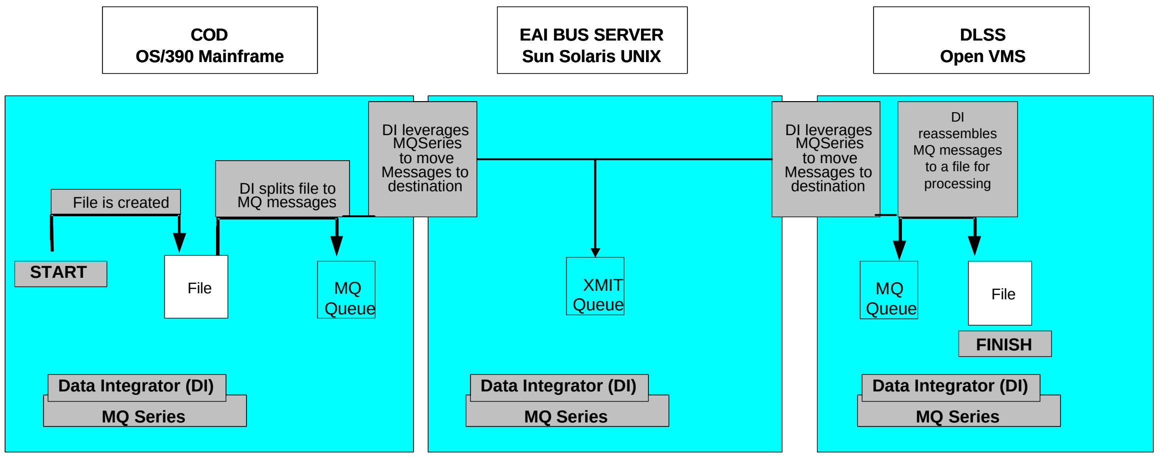
Figure 3: COD – DLSS Interfaces

NOTE: EAI Bus has been abstracted out

2.7.4.1 DLSS Batch Feed Detailed Design Description

The sample function selected for the DLSS Batch Feed interface validates the ability to send COD data from the send script to the DLSS system. Once the source file is completed (process is done writing to it), Data Integrator will move the flat file from COD to the appropriate destination location in the DLSS system via MQSeries queues and channels.

The figure below describes the message flow for the DLSS Batch Feed interface.


COD
OS/390 Mainframe
EAI BUS SERVER
Sun Solaris UNIX
DLSS
Open VMS
File is created
START
File
DI splits file to MQ messages
MQ Queue
DI leverages MQSeries to move Messages to destination
XMIT Queue
DI leverages MQSeries to move Messages to destination
DI reassembles MQ messages to a file for processing
MQ Queue
File
FINISH
Data Integrator (DI)
MQ Series

The flow of a bulk file from COD to DLSS, via Data Integrator and MQSeries queues is as follows:

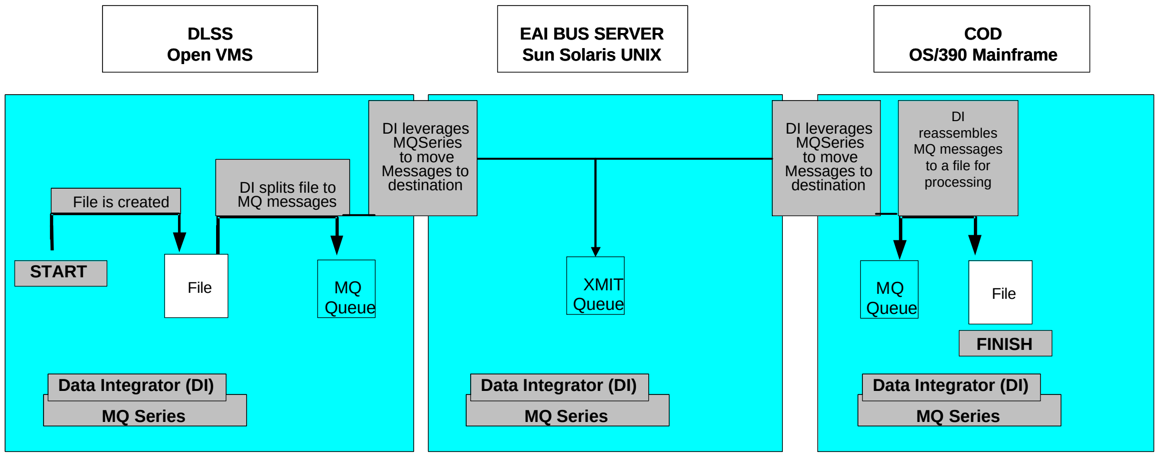
1. A file is created in a specified directory.

2. A JCL is called to Data Integrator sender adapter to initiate transfer of the file.
3. The Sender adapter splits the file into MQ messages, and the source (local) Queue Manager (VD0Q) on COD moves the message to the specified XMIT queue (EAIA1).
4. The message is transmitted directly to the destination Receiver (Queue Manager DLSST1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (DLSST1.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. The source file is deleted on a successful file transfer to DLSS.

2.7.4.2 DLSS Batch Response Detailed Design Description

The sample function selected for the DLSS Batch Response interface validates the ability to send DLSS data from the send script to the COD system. Once the source file is completed (process is done writing to it), Data Integrator moves the flat file from DLSS to the appropriate destination location in the COD system via MQSeries queues and channels.

The figure below describes the message flow for the DLSS Batch Response interface.



The flow of a bulk file from DLSS to COD, via Data Integrator and MQSeries queues is as follows:

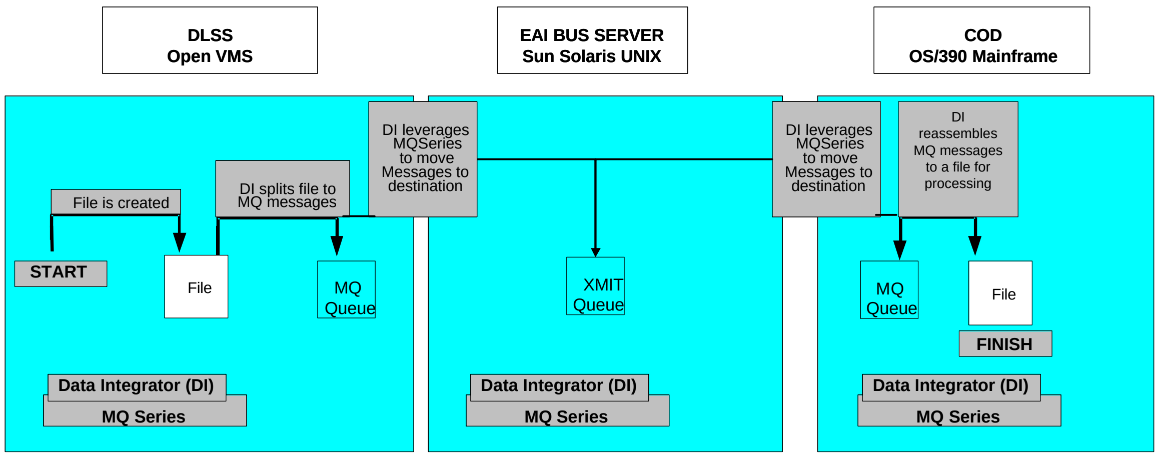
1. A file is created in a specified directory.
2. An Open VMS step is called to Data Integrator sender adapter to initiate transfer of the file.
3. The Sender adapter splits the file into MQ messages, and the source (local) Queue Manager (DLSST1) on DLSS moves the message to the specified XMIT queue (EAIA1).

4. The message is transmitted directly to the destination Receiver (Queue Manager VD0Q). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (VD0Q.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. The source file is deleted on a successful file transfer to COD.

2.7.4.3 Disbursement Confirmations Detailed Design Description

The sample function selected for the Disbursement Confirmations interface validates the ability to send DLSS data from the send script to the COD system. Once the source file is completed (process is done writing to it), Data Integrator moves the flat file from DLSS to the appropriate destination location in the COD system via MQSeries queues and channels.

The figure below describes the message flow for the Disbursement Confirmations interface.



The flow of a bulk file from DLSS to COD, via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. An Open VMS step is called to Data Integrator sender adapter to initiate transfer of the file.
3. The Sender adapter splits the file into MQ messages, and the source (local) Queue Manager (DLSST1) on DLSS moves the message to the specified XMIT queue (EAIA1).
4. The message is transmitted directly to the destination Receiver (Queue Manager VD0Q). If the Receiver is on a remote queue manager (in this case), the message destination is

resolved to a transmission queue or set of transmission queues (VD0Q.B) and appropriately transmitted via the MQSeries message channel agent.

5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. The source file is deleted on a successful file transfer to COD.

2.7.5 COD to FMS Interface Overview

All financial transactions, the actual movement of funding authority, between Schools and FSA are managed through the FSA Financial Management System. Schools will have the ability to draw cash from Treasury through FMS draw down capability, or, if that capability is not yet in place, through the Government Accounts Payable System (GAPS). The interface between COD and FMS will be the same regardless of whether or not GAPS is involved in the funds disbursement or processing.

FMS and COD share information in three categories: 1) Financial Transactions, 2) School Information, and 3) Reconciliation and Balancing.

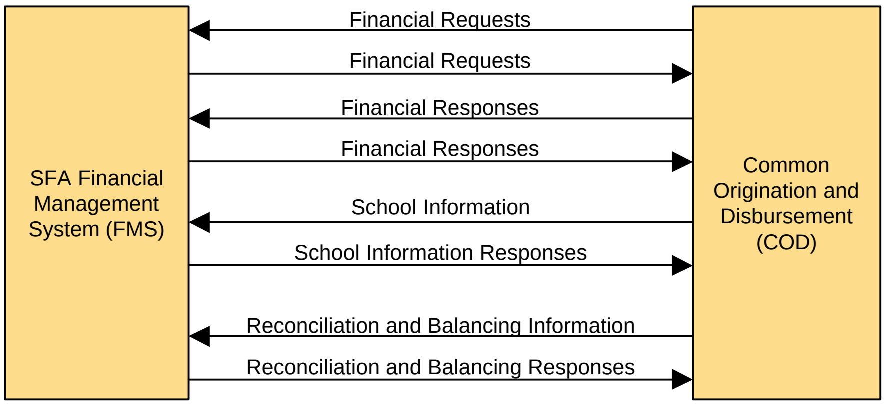
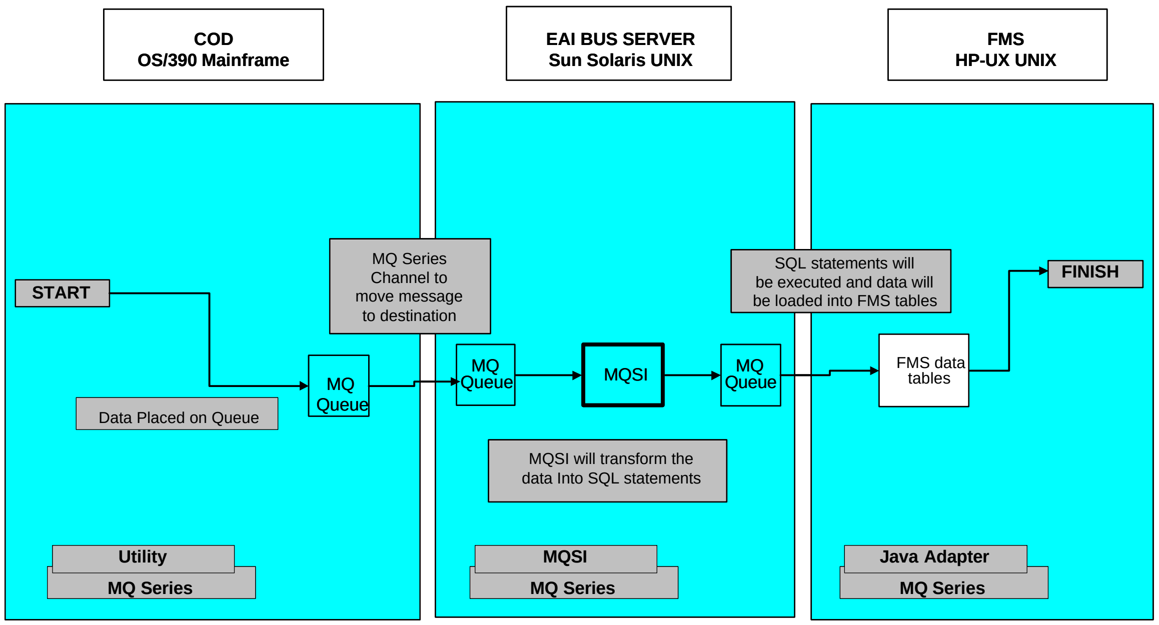

SFA Financial Management System (FMS)
Common Origination and Disbursement (COD)
Financial Requests
Financial Requests
Financial Responses
Financial Responses
School Information
School Information Responses
Reconciliation and Balancing Information
Reconciliation and Balancing Responses

Figure 4: COD – FMS Interfaces

NOTE: EAI Bus has been abstracted out

2.7.5.1 Financial Transactions Input Detailed Design Description

The sample function selected for the Financial Transactions Input interface validates the ability to send COD data to the FMS system in real time. Once the COD data is placed on the queue, the channel moves the message from the COD queue to the FMS queue. On the EAI Bus, MQSI transforms the messages into SQL statements and the data is loaded into FMS data tables.



The figure below describes the message flow for the Financial Transactions Input interface.

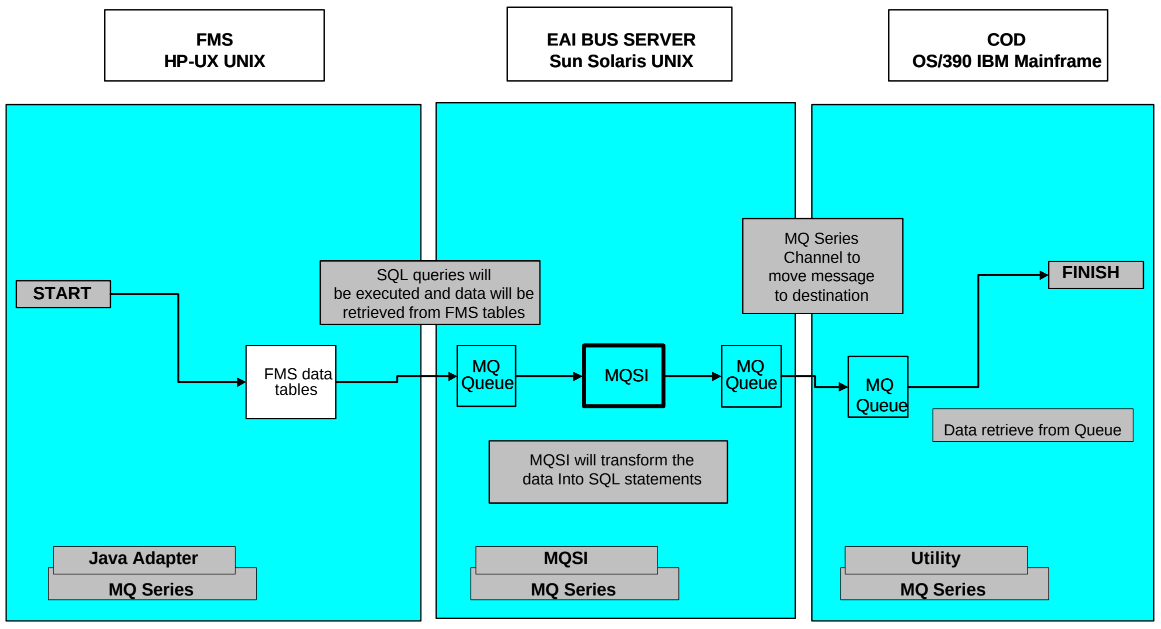
The flow of a transactional real time message from COD to FMS, via MQSeries queue and channel is as follows:

1. A Financial Transaction is created.
2. The Sender adapter (COD Utility) converts the data into MQ messages, and the source (local) Queue Manager (VD0Q) on COD moves the message to the specified XMIT queue (EAIT1).
3. The message is transmitted directly to the destination Receiver (Queue Manager FMST1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (FMST1.B) and appropriately transmitted via the MQSeries message channel agent.
4. The Sender replies to the originating Manager, indicating that the file has been submitted.
5. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
6. The Receiver adapter transforms the data into one MQ message and initiates an MQSI message flow. An operational reply is submitted to the originating Manager.
7. MQSI transforms each line of data into an SQL statement and passes the SQL statement as a message to a queue.
8. Using a custom Java adapter, the SQL message is executed against the FMS Oracle database and the data is loaded into the appropriate tables.

2.7.5.2 Financial Transactions Retrieval Detailed Design Description

The sample function selected for the Financial Transactions Retrieval interface validates the ability to send FMS data to the COD system in real time. Once the FMS data is placed on the queue, the channel moves the message from the FMS queue to the COD queue. On the EAI Bus, MQSI transforms the SQL queries into COD readable messages and COD Utility retrieves the messages.

The figure below describes the message flow for the Financial Transactions Retrieval interface.



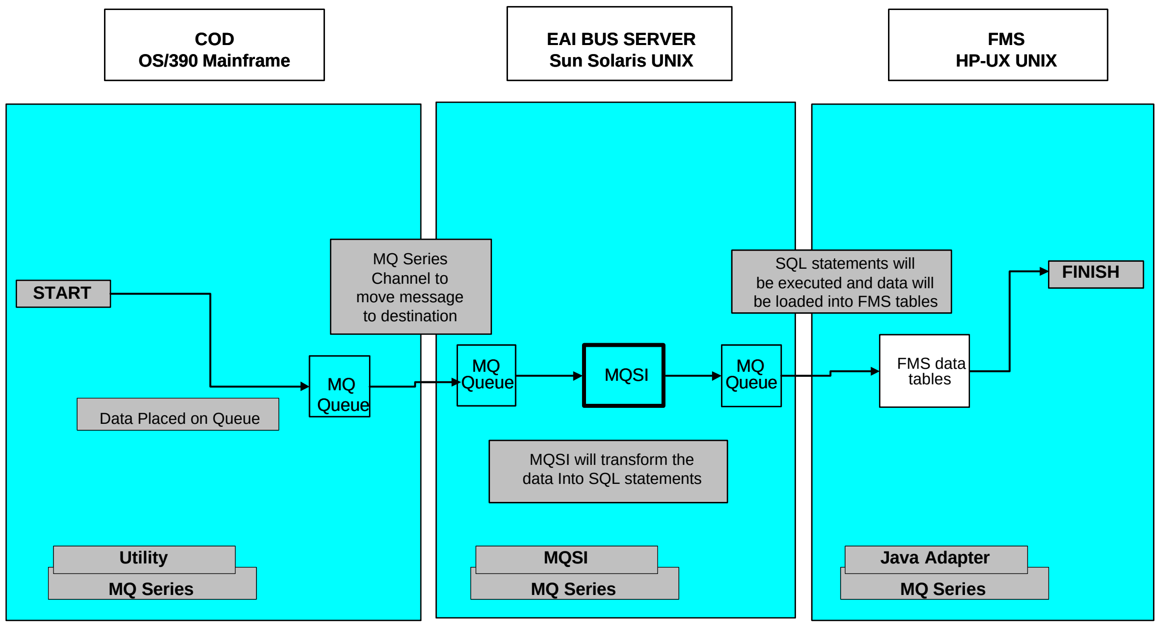
The flow of a transactional real time message from FMS to COD, via MQSeries queue and channel is as follows:

1. A Financial Transaction Retrieval is requested as SQL queries.
2. The Sender adapter (FMS Java adaptor) converts the data into MQ messages, and the source (local) Queue Manager (FMST1) on FMS moves the message to the specified XMIT queue (EAIT1).
3. The message is transmitted directly to the destination Receiver (Queue Manager VD0Q). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (VD0Q.B) and appropriately transmitted via the MQSeries message channel agent.
4. The Sender replies to the originating Manager, indicating that the file has been submitted.
5. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
6. The Receiver adapter transforms the data into one MQ message and initiates an MQSI message flow. An operational reply is submitted to the originating Manager.
7. MQSI transforms each line of data from an SQL statement to a COD readable message and passes the message to a queue.
8. The COD Utility retrieves the message off the queue and processes the message.

2.7.5.3 School Information Input Detailed Design Description

The sample function selected for the Financial Transactions Input interface validates the ability to send COD data to the FMS system in real time. Once the COD data is placed on the queue, the channel moves the message from the COD queue to the FMS queue. On the EAI Bus, MQSI transforms the messages into SQL statements and the data is then loaded into FMS data tables.

The figure below describes the message flow for the School Information Input interface.



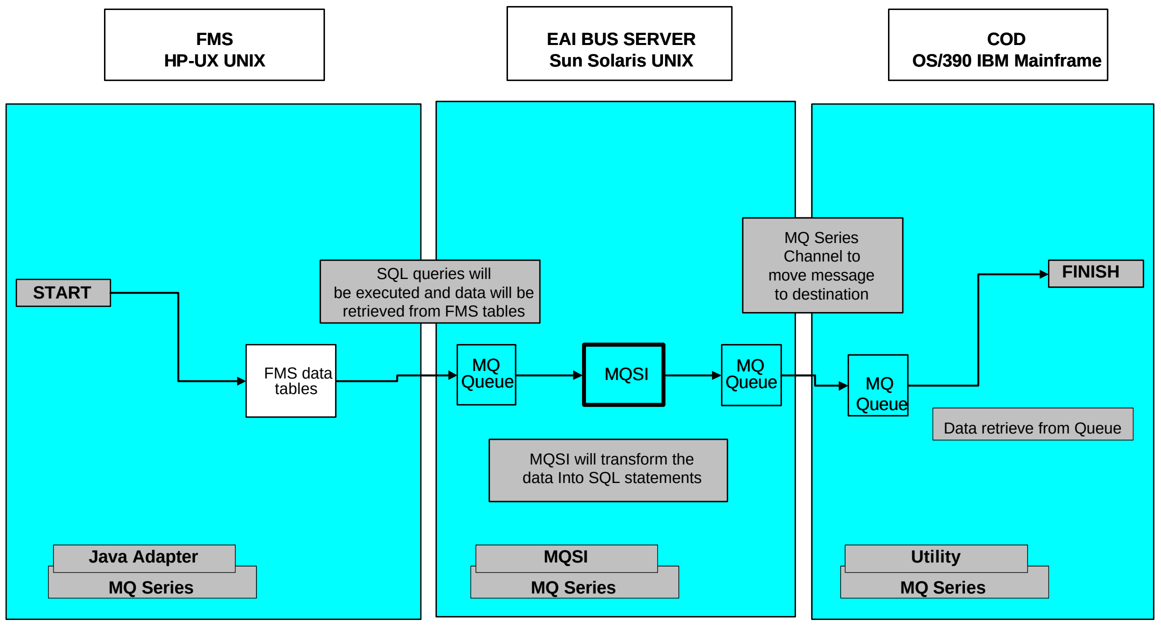
The flow of a transactional real time message from COD to FMS, via MQSeries queue and channel is as follows:

1. A School Information is created.
2. The Sender adapter (COD Utility) converts the data into MQ messages, and the source (local) Queue Manager (VD0Q) on COD moves the message to the specified XMIT queue (EAIT1).
3. The message is transmitted directly to the destination Receiver (Queue Manager FMST1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (FMST1.B) and appropriately transmitted via the MQSeries message channel agent.
4. The Sender replies to the originating Manager, indicating that the file has been submitted.
5. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
6. The Receiver adapter transforms the data into one MQ message and initiates an MQSI message flow. An operational reply is submitted to the originating Manager.
7. MQSI transforms each line of data into an SQL statement and passes the SQL statement as a message to a queue.
8. Using a custom Java adapter, the SQL message is executed against the FMS Oracle database and the data is loaded into the appropriate tables.

2.7.5.4 School Information Retrieval Detailed Design Description

The sample function selected for the School Information Retrieval interface validates the ability to send FMS data to the COD system in real time. Once the FMS data is placed on the queue, the channel moves the message from the FMS queue to the COD queue. On the EAI Bus, MQSI transforms the SQL queries into COD readable messages and COD Utility retrieves the messages.

The figure below describes the message flow for the School Information Retrieval interface.



The flow of a transactional real time message from FMS to COD, via MQSeries queue and channel is as follows:

1. A School Information Retrieval is requested as SQL queries.
2. The Sender adapter (FMS Java adaptor) converts the data into MQ messages, and the source (local) Queue Manager (FMST1) on FMS moves the message to the specified XMIT queue (EAIT1).
3. The message is transmitted directly to the destination Receiver (Queue Manager VD0Q). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (VD0Q.B) and appropriately transmitted via the MQSeries message channel agent.
4. The Sender replies to the originating Manager, indicating that the file has been submitted.
5. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
6. The Receiver adapter transforms the data into one MQ message and initiates an MQSI message flow. An operational reply is submitted to the originating Manager.
7. MQSI transforms each line of data from an SQL statement to a COD readable message and passes the message to a queue.
8. The COD Utility retrieves the message off the queue and processes the message.

2.7.6 COD to NSLDS Interface Overview

The National Student Loan Data System (NSLDS) is a national database of loans and other financial aid disbursed to students under Title IV of the Higher Education Act of 1964. It supports the entire student aid community in a variety of operational and research functions for improving the administration of Title IV aid programs. Enrolling in the NSLDS is required of all institutions that participate in Title IV federal student financial aid programs. The NSLDS includes data on the FFEL, Direct Loan, and Perkins Loan programs; on Pell awards and disbursements; and on Pell and FSEOG over awards.

NSLDS receives data from multiple internal and external sources to the Department of Education, and maintains the data in one integrated database. This data is available to many different system users for administration, research support, policy analysis, and other management purposes.

To help schools determine if a student is in default or owes a repayment, the CPS matches applications with the NSLDS database. Schools are responsible for reconciling all information received about a student before disbursing aid. Therefore, schools are required to resolve any conflicts between the NSLDS information and information the student has provided. For example, if the NSLDS indicates that a student is not in default but the school has documentation indicating that the student is in default, the school must resolve this conflict before disbursing federal student aid.

COD provides NSLDS with data about Pell Grant disbursements and eligibility via the Pell Recipient Information file. In return, NSLDS provides COD with errors related to that file. COD will not send NSLDS loan disbursement processing and eligibility, as that information will be sent to NSLDS from Loan Servicing (DLSS).

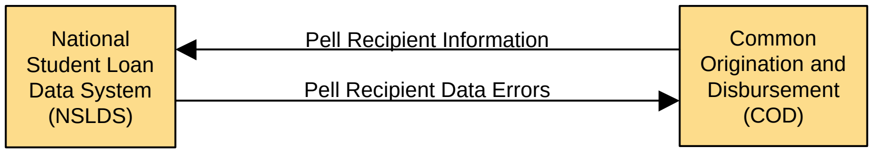

National Student Loan Data System (NSLDS)
Pell Recipient Information
Pell Recipient Data Errors
Common Origination and Disbursement (COD)

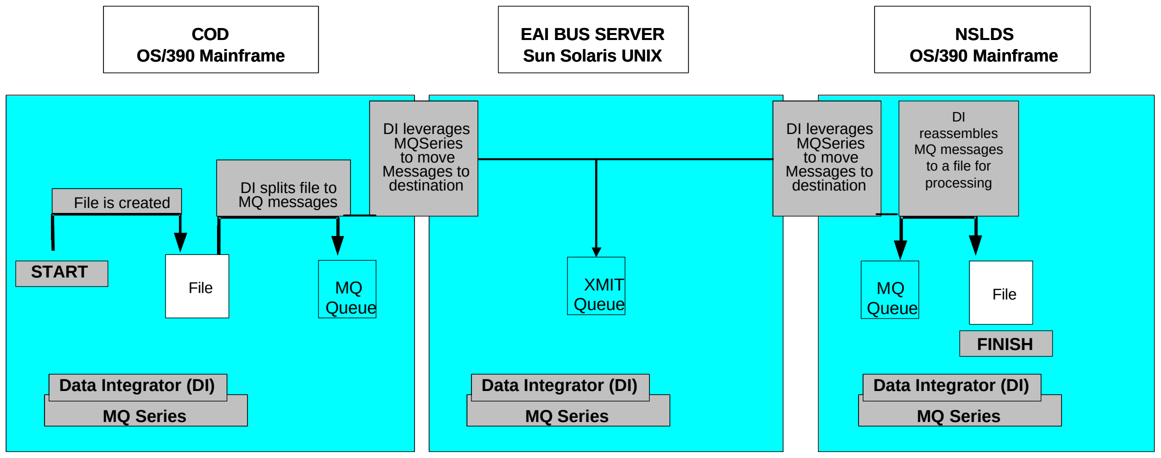
Figure 5: COD – NSLDS Interfaces

NOTE: EAI Bus has been abstracted out

2.7.6.1 Pell Recipient Information Detailed Design Description

The sample function selected for the Pell Recipient Information interface validates the ability to send COD data from the send script to the NSLDS system. Once the source file is completed (process is done writing to it), Data Integrator moves the flat file from COD to the appropriate destination location in the NLSDS system via MQSeries queues and channels.

The figure below describes the message flow for the Pell Recipient Information Interface.



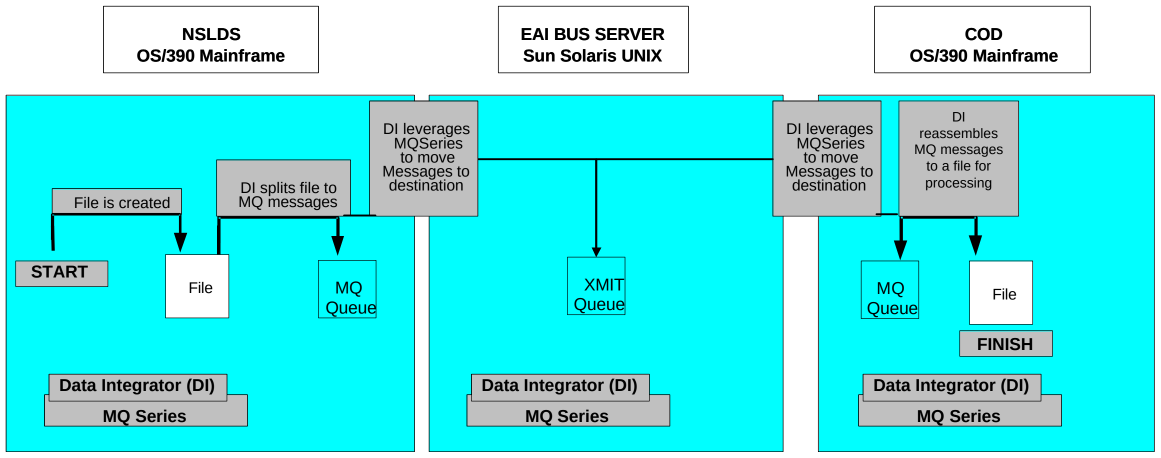
The flow of a bulk file from COD to NSLDS, via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. A JCL is called to Data Integrator sender adapter to initiate transfer of the file.
3. The Sender adapter splits the file into MQ messages, and the source (local) Queue Manager (VD0Q) on COD moves the message to the specified XMIT queue (EAIA1).
4. The message is transmitted directly to the destination Receiver (Queue Manager QNT1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (QNT1.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. Post-Processing is kicked off upon receipt of file on NSLDS.
9. The source file is deleted on a successful file transfer to NSLDS.

2.7.6.2 Pell Recipient Data Errors Detailed Design Description

The sample function selected for the Pell Recipient Data Errors interface validates the ability to send NSLDS data from the send script to the COD system. Once the source file is completed (process is done writing to it), Data Integrator will move the flat file from NSLDS to the appropriate destination location in the COD system via MQSeries queues and channels.

The figure below describes the message flow for the Pell Recipient Data Errors Interface.



The flow of a bulk file from NSLDS to COD, via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. A JCL is called to Data Integrator sender adapter to initiate transfer of the file.
3. The Sender adapter splits the file into MQ messages, and the source (local) Queue Manager (QNT1) on NSLDS moves the message to the specified XMIT queue (EAIA1).
4. The message is transmitted directly to the destination Receiver (Queue Manager VD0Q). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (VD0Q.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. Post-Processing is kicked off upon receipt of file on COD.
9. The source file is deleted on a successful file transfer to COD.

2.7.7 COD to PEPS Interface Overview

The Postsecondary Education Participants System (PEPS) is the Federal Student Aid (FSA) management information system of all organizations that have a role in administering Student Financial Aid and other Higher Education Act programs. PEPS maintains eligibility, certification, demographic, review, audit and default rate data on all Schools, Lenders, and Guarantors participating in the Title IV programs.

PEPS provides COD with data about Schools and School eligibility in the Daily Participants Feed.

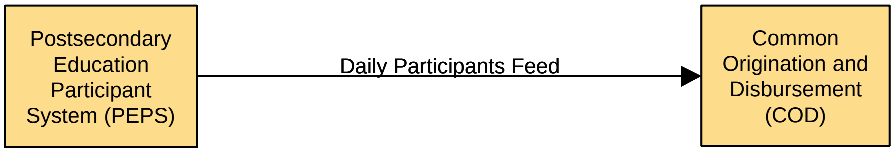

Postsecondary Education Participant System (PEPS)
Daily Participants Feed
Common Origination and Disbursement (COD)

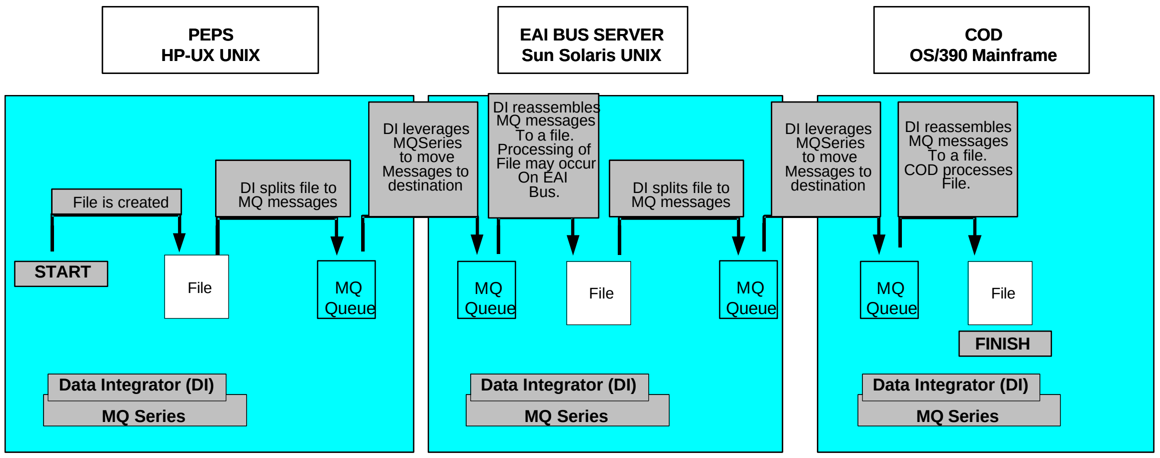
Figure 6: COD – PEPS Interface

NOTE: EAI Bus has been abstracted out

2.7.7.1 Daily Participants Feed Detailed Design Description

The sample function selected for the Daily Participants Feed interface validates the ability to send PEPS data using the directory monitoring service to initiate the file transfer to COD. The EAI team configured Data Integrator's directory monitoring service to watch for the complete creation of the Daily Participants Feed file in a specified directory. The directory monitoring process looks at a specific directory for the creation or revision of any files that fit the naming pattern that it has been configured to look for. Once the source file is completed (process is done writing to it), Data Integrator will move the flat file from PEPS to the EAI Bus, extract the "delta" from the Daily Participant file and move the PEPS "delta" file to the appropriate destination location in the COD system via MQSeries queues and channels.

The figure below describes the message flow for the Daily Participants Feed interface.


PEPS HP-UX UNIX
EAI BUS SERVER Sun Solaris UNIX
COD OS/390 Mainframe
File is created
START
File
DI splits file to MQ messages
MQ Queue
DI leverages MQSeries to move Messages to destination
DI reassembles MQ messages To a file. Processing of File may occur On EAI Bus.
MQ Queue
File
DI splits file to MQ messages
MQ Queue
DI leverages MQSeries to move Messages to destination
DI reassembles MQ messages To a file. COD processes File.
MQ Queue
File
FINISH
Data Integrator (DI)
MQ Series
Data Integrator (DI)
MQ Series
Data Integrator (DI)
MQ Series

The flow of a bulk file from PEPS to COD via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. The dirmon process uses the Data Integrator sender adapter to initiate transfer of the file.
3. The sender adapter splits the file into MQ messages, and the source (local) Queue Manager (PEPST1) on PEPS moves the message to the specified XMIT queue.
4. The message is transmitted to the Receiver (EAI Queue Manager EAIA1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (EAIA1.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. The source file is deleted on a successful file transfer to the EAI bus.
9. Upon successful file transfer to the bus, Data Integrator calls a Java transformation program to extract the PEPS "delta" file from the complete PEPS file.
10. Upon successful extract of the PEPS "delta" file, Data Integrator calls a script to initiate transfer of the file to COD.
11. The sender adapter splits the file into MQ messages, and the source (local) Queue Manager (EAIA1) on the EAI bus moves the message to the specified XMIT queue.
12. The message is transmitted to the Receiver (COD Queue Manager VD0Q). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (VD0Q.B) and appropriately transmitted via the MQSeries message channel agent.
13. The Sender replies to the originating Manager, indicating that the file has been submitted.
14. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
15. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
16. Upon successful file transfer to COD, Data Integrator calls a post processor script to handle the Daily Participants Feed.

Files

The following input files have been defined on PEPS for access by the EAI Core test application in execution of this test scenario:

Filenames	Function
schfile_extract_daily_yyyymmdd.Z	Compressed PEPS school data file

2.7.8 COD to SAIG Interface Overview

The SAIG Secure Portal writes the School information to the COD mailbox. This interface enables the schools (via bTrade) to send and receive data to and from COD. Common Records and Legacy Records sent from the schools are directly processed at COD. Common Records, Common Record acknowledgements, and Common Record responses are XML based, whereas Legacy Records are Legacy based. COD only sends out XML based Common Record acknowledgements and responses to the EAI Bus. Common Record acknowledgements and responses must be transformed before submitting to Common Record schools' mailbox and Legacy Record schools' mailbox.

This interface allows COD to retrieve a batch of input records from SAIG. Files transported from SAIG to COD include Common Record Files, Converted Pell and Direct Loan Legacy Files, and a School Information File. Files transported from COD to SAIG include converted Legacy Record Responses, Common Record Responses, and Reports.

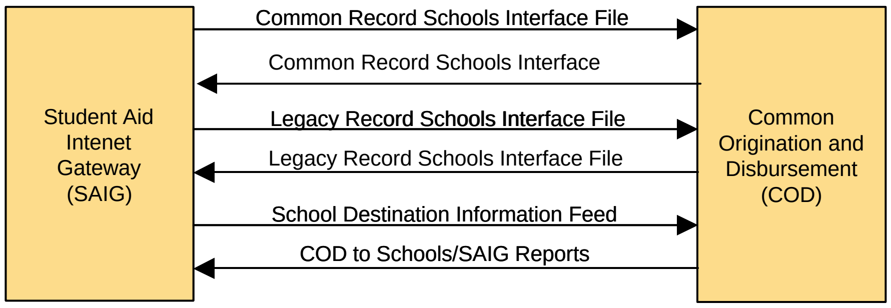
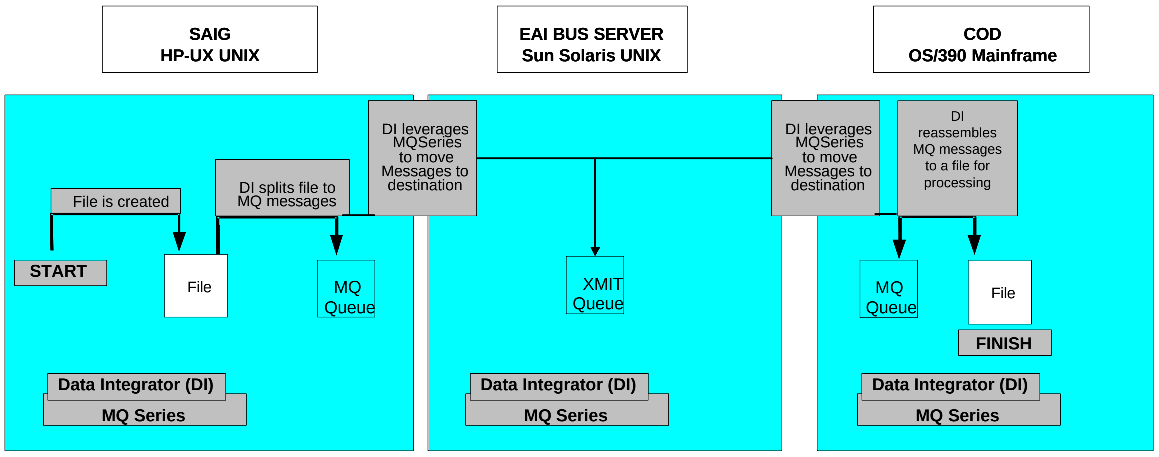

Student Aid Intenet Gateway (SAIG)
Common Record Schools Interface File
Common Record Schools Interface
Legacy Record Schools Interface File
Legacy Record Schools Interface File
School Destination Information Feed
COD to Schools/SAIG Reports
Common Origination and Disbursement (COD)

Figure 7: COD – SAIG Interfaces

NOTE: EAI Bus has been abstracted out

2.7.8.1 Common Record Input Detailed Design Description

The sample function selected for the Common Record Input interface validates the ability to send SAIG data from the send script to the COD system. Once the source file is completed (process is done writing to it), Data Integrator will move the XML file from SAIG to the appropriate destination location in the COD system via MQSeries queues and channels.



The figure above describes the message flow for the Common Record Input Interface.

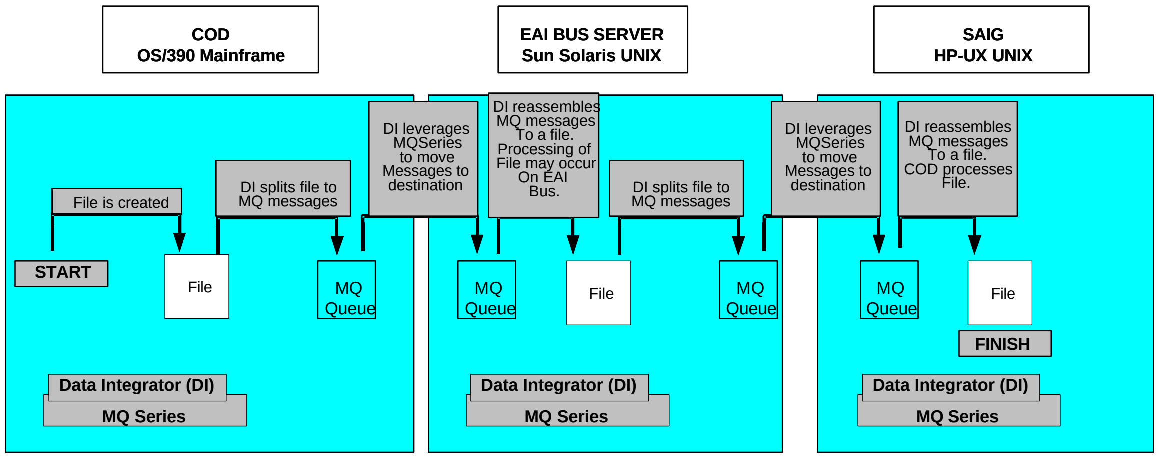
The flow of a bulk file from SAIG to COD, via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. COD polls SAIG mailbox that calls Data Integrator sender adapter to initiate transfer of the file.
3. The Sender adapter splits the file into MQ messages, and the source (local) Queue Manager (SAIGT1) on SAIG moves the message to the specified XMIT queue (EAIA1).
4. The message is transmitted directly to the destination Receiver (Queue Manager VD0Q). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (VD0Q.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. Post-Processing is kicked off upon receipt of file on COD.
9. The source file is deleted on a successful file transfer to COD.

2.7.8.2 Common Record Acknowledgements and Responses Detailed Design Description

The sample function selected for the Common Record Acknowledgements and Responses interface validates the ability to send COD acknowledgements and responses to the SAIG system from the send scripts. Once the source file is completed (process is done writing to it), Data Integrator moves the XML files to the appropriate destination location in the SAIG system via MQSeries queues and channels.

The figures below describe the message flow for the Common Record Acknowledgements and Responses interface.


COD
OS/390 Mainframe
EAI BUS SERVER
Sun Solaris UNIX
SAIG
HP-UX UNIX
File is created
DI splits file to MQ messages
DI leverages MQSeries to move Messages to destination
DI reassembles MQ messages To a file. Processing of File may occur On EAI Bus.
DI splits file to MQ messages
DI leverages MQSeries to move Messages to destination
DI reassembles MQ messages To a file. COD processes File.
START
File
MQ Queue
MQ Queue
File
MQ Queue
MQ Queue
File
FINISH
Data Integrator (DI)
MQ Series
Data Integrator (DI)
MQ Series
Data Integrator (DI)
MQ Series

The flow of a bulk file from COD to SAIG via Data Integrator and MQSeries queues is as follows:

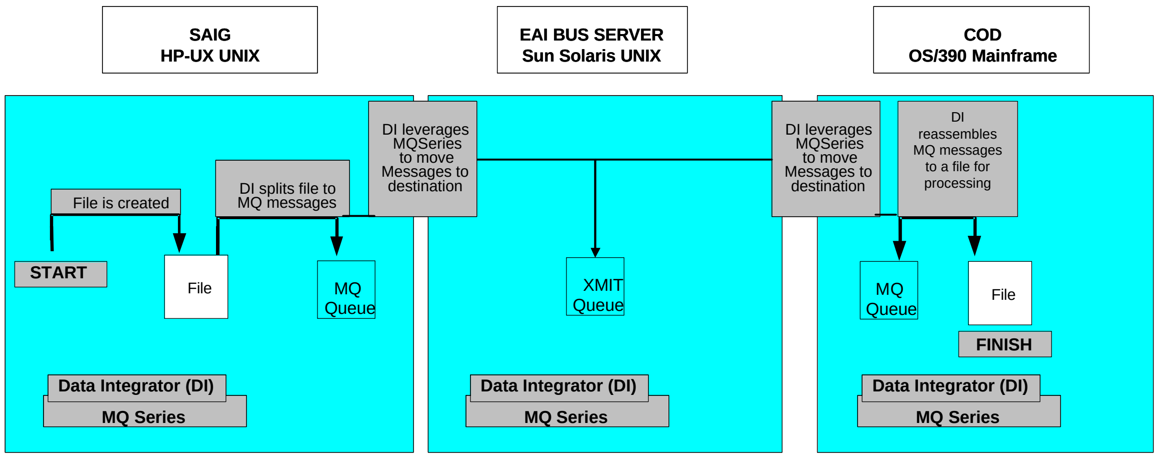
1. Once COD receives the Common Record file, COD creates a Common Record Acknowledgement file in a specified directory. After COD processes the Common Record file and validates its contents, COD creates a Common Record Response file in a specified directory.
2. A JCL is called to Data Integrator sender adapter to initiate transfer of the file.
3. The sender adapter splits the file into MQ messages, and the source (local) Queue Manager (VD0Q1) on COD moves the message to the specified XMIT queue.
4. The message is transmitted to the Receiver (EAI Queue Manager EAIA1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (EAIA1.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. The source file is deleted on a successful file transfer to the EAI bus.

9. Upon successful file transfer to the bus, Data Integrator calls a Java transformation program to convert the Common Record Acknowledgement and Response files to XML formatted files.
10. Upon successful conversion to Common Record files, Data Integrator calls a script to initiate transfer of the files to the appropriate SAIG schools' mailbox.
11. The sender adapter splits the file into MQ messages, and the source (local) Queue Manager (EAIA1) on the EAI bus moves the message to the specified XMIT queue.
12. The message is transmitted to the Receiver (SAIG Queue Manager SAIGT1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (SAIGT1.B) and appropriately transmitted via the MQSeries message channel agent.
13. The Sender replies to the originating Manager, indicating that the file has been submitted.
14. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
15. The Receiver reassembles the MQ message to a XML file and submits and operational reply to the originating Manager.
16. Upon successful file transfer to SAIG, Data Integrator calls a post processor script to handle the Common Record files.

2.7.8.3 Legacy Record Input Detailed Design Description

The sample function selected for the Legacy Record Input interface validates the ability to send SAIG data from the send script to the COD system. Once the source file is completed (process is done writing to it), Data Integrator moves the flat file from SAIG to the appropriate destination location in the COD system via MQSeries queues and channels.

The figure below describes the message flow for the Legacy Record Input Interface.



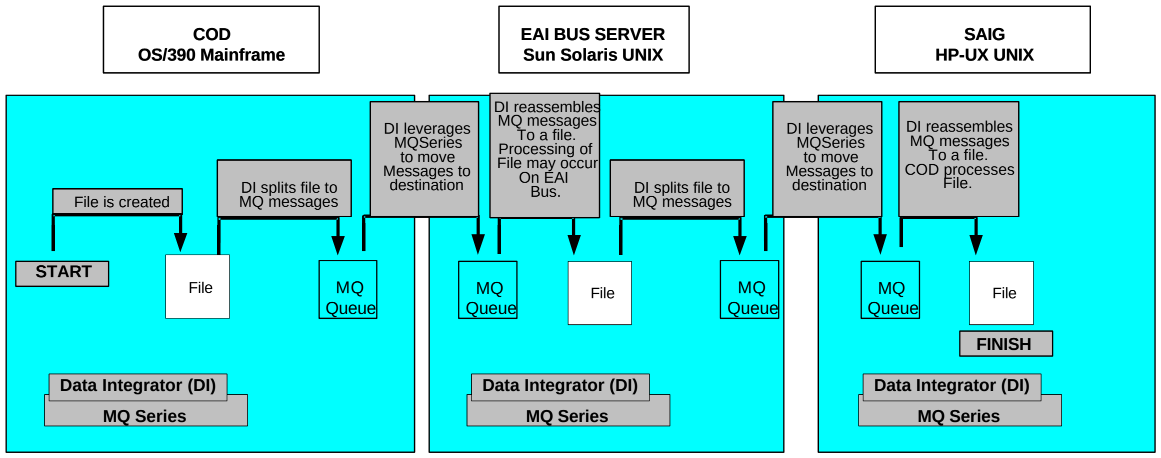
The flow of a bulk file from SAIG to COD, via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. COD polls SAIG mailbox that calls Data Integrator sender adapter to initiate transfer of the file.
3. The Sender adapter splits the file into MQ messages, and the source (local) Queue Manager (SAIGT1) on SAIG moves the message to the specified XMIT queue (EAIA1).
4. The message is transmitted directly to the destination Receiver (Queue Manager VD0Q). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (VD0Q.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. Post-Processing is kicked off upon receipt of file on COD.
9. The source file is deleted on a successful file transfer to COD.

2.7.8.4 Legacy Record Acknowledgements and Responses Detailed Design Description

The sample function selected for the Legacy Record Acknowledgements and Responses interface validates the ability to send COD acknowledgements and responses to the SAIG system from the send scripts. Once the source file is completed (process is done writing to it), Data Integrator will move the XML files to the appropriate destination location in the SAIG system via MQSeries queues and channels.

The figures below describe the message flow for the Legacy Record Acknowledgements and Responses interface.


COD
OS/390 Mainframe
EAI BUS SERVER
Sun Solaris UNIX
SAIG
HP-UX UNIX
File is created
START
File
DI splits file to MQ messages
MQ Queue
DI leverages MQSeries to move Messages to destination
DI reassembles MQ messages To a file. Processing of File may occur On EAI Bus.
MQ Queue
File
DI splits file to MQ messages
MQ Queue
DI leverages MQSeries to move Messages to destination
DI reassembles MQ messages To a file. COD processes File.
MQ Queue
File
FINISH
Data Integrator (DI)
MQ Series
Data Integrator (DI)
MQ Series
Data Integrator (DI)
MQ Series

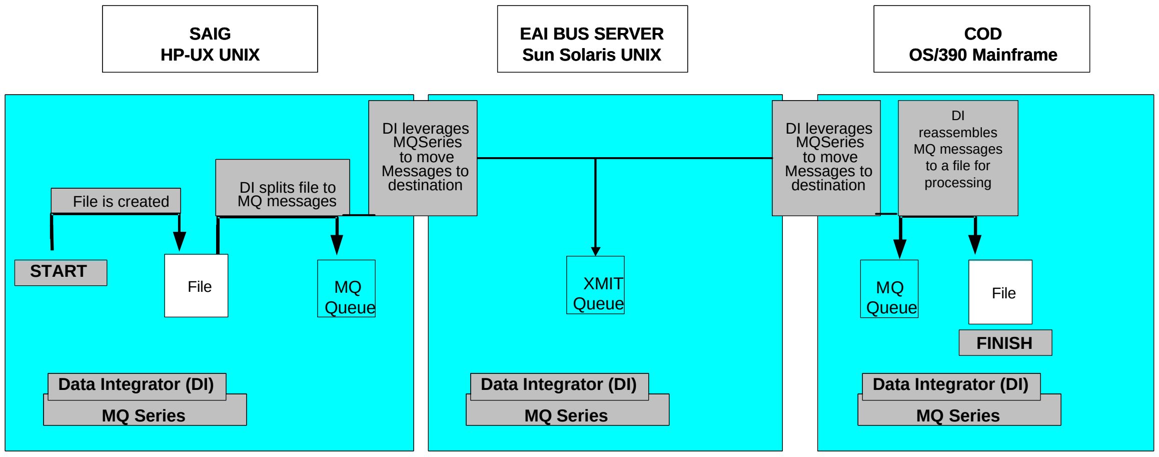
The flow of a bulk file from COD to SAIG via Data Integrator and MQSeries queues is as follows:

1. Once COD receives the Legacy Record file, COD creates a XML based Common Record Acknowledgement file in a specified directory. After COD processes the Legacy Record file and validates its contents, COD creates a XML based Common Record Response file in a specified directory.
2. A JCL is called to Data Integrator sender adapter to initiate transfer of the file.
3. The sender adapter splits the file into MQ messages, and the source (local) Queue Manager (VD0Q1) on COD moves the message to the specified XMIT queue.
4. The message is transmitted to the Receiver (EAI Queue Manager EAIA1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (EAIA1.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. The source file is deleted on a successful file transfer to the EAI bus.
9. Upon successful file transfer to the bus, Data Integrator calls a Java transformation program to convert the Common Record Acknowledgement and Response files to Legacy formatted files.
10. Upon successful conversion to Legacy Record files, Data Integrator calls a script to initiate transfer of the files to the appropriate SAIG schools' mailbox.
11. The sender adapter splits the file into MQ messages, and the source (local) Queue Manager (EAIA1) on the EAI bus moves the message to the specified XMIT queue.
12. The message is transmitted to the Receiver (SAIG Queue Manager SAIGT1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (SAIGT1.B) and appropriately transmitted via the MQSeries message channel agent.
13. The Sender replies to the originating Manager, indicating that the file has been submitted.
14. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
15. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
16. Upon successful file transfer to SAIG, Data Integrator calls a post processor script to handle the Legacy Record files.

2.7.8.5 COD-SAIG School Destination Information Feed Detailed Design Description

The sample function selected for the SAIG to COD interface validates the ability to send SAIG data from the send script to the COD system. Once the source file is completed (process is done writing to it), Data Integrator moves the flat file from SAIG to the appropriate destination location in the COD system via MQSeries queues and channels.

The figure below describes the message flow for the School Destination Information Feed Interface.



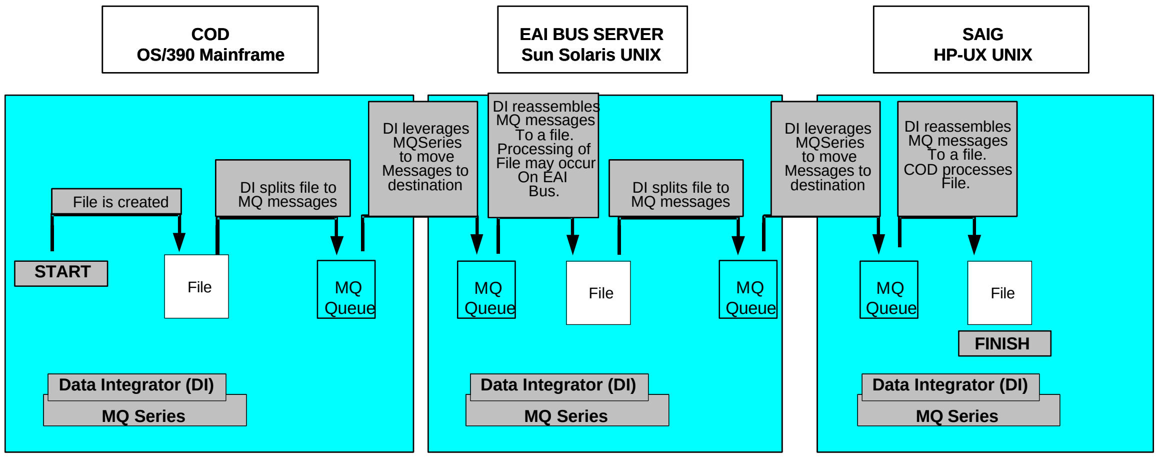
The flow of a bulk file from SAIG to COD, via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. COD polls SAIG mailbox that calls Data Integrator sender adapter to initiate transfer of the file.
3. The Sender adapter splits the file into MQ messages, and the source (local) Queue Manager (SAIGT1) on SAIG moves the message to the specified XMIT queue (EAIA1).
4. The message is transmitted directly to the destination Receiver (Queue Manager VD0Q). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (VD0Q.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. Post-Processing is kicked off upon receipt of file on COD.
9. The source file is deleted on a successful file transfer to COD.

2.7.8.6 COD to Schools/SAIG Reports Detailed Design Description

The sample function selected for the COD to Schools/SAIG Reports interface validates the ability to send COD data from the send script to the SAIG system. Once the source file is completed (process is done writing to it), Data Integrator moves the flat file from COD to the appropriate destination location in the SAIG system via MQSeries queues and channels.

The figure below describes the message flow for the COD to Schools/SAIG Reports interface.


COD
OS/390 Mainframe
EAI BUS SERVER
Sun Solaris UNIX
SAIG
HP-UX UNIX
File is created
DI splits file to MQ messages
DI leverages MQSeries to move Messages to destination
DI reassembles MQ messages To a file. Processing of File may occur On EAI Bus.
DI splits file to MQ messages
DI leverages MQSeries to move Messages to destination
DI reassembles MQ messages To a file. COD processes File.
START
File
MQ Queue
MQ Queue
File
MQ Queue
MQ Queue
File
FINISH
Data Integrator (DI)
MQ Series
Data Integrator (DI)
MQ Series
Data Integrator (DI)
MQ Series

The flow of a bulk file from COD to SAIG via Data Integrator and MQSeries queues is as follows:

1. A file is created in a specified directory.
2. The JCL is called to Data Integrator sender adapter to initiate transfer of the file.
3. The sender adapter splits the file into MQ messages, and the source (local) Queue Manager (VD0Q) on COD moves the message to the specified XMIT queue.
4. The message is transmitted to the Receiver (EAI Queue Manager EAIA1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (EAIA1.B) and appropriately transmitted via the MQSeries message channel agent.
5. The Sender replies to the originating Manager, indicating that the file has been submitted.
6. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
7. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
8. The source file is deleted on a successful file transfer to the EAI bus.
9. Upon successful file transfer to the bus, Data Integrator calls a Java transformation program to separate the files for each school.
10. Upon successful separation of the files, Data Integrator calls a script to initiate transfer of the files to the appropriate SAIG schools' mailbox.

11. The sender adapter splits the file into MQ messages, and the source (local) Queue Manager (EAIA1) on the EAI bus moves the message to the specified XMIT queue.
12. The message is transmitted to the Receiver (SAIG Queue Manager SAIGT1). If the Receiver is on a remote queue manager (in this case), the message destination is resolved to a transmission queue or set of transmission queues (SAIGT1.B) and appropriately transmitted via the MQSeries message channel agent.
13. The Sender replies to the originating Manager, indicating that the file has been submitted.
14. The Receiver adapter receives data from MQSeries to create the target data. The receiver accepts a data transfer request and processes the inbound data from its data queues.
15. The Receiver reassembles the MQ message to a flat file and submits an operational reply to the originating Manager.
16. Upon successful file transfer to SAIG, Data Integrator calls a post processor script to handle the COD to Schools/SAIG Reports.

3 EAI TEST METHODOLOGY

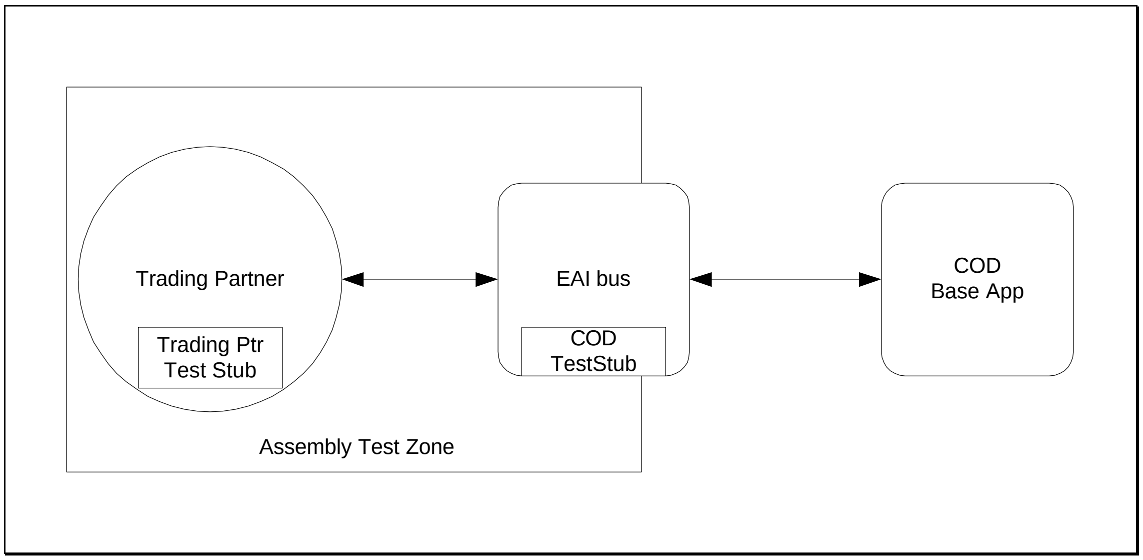
3.1 Interface Testing Process

For the purposes of this document, only the Interface Assembly Testing process will be detailed. The purpose of Interface Assembly Testing is to validate that the construction of Release 3.0 application interfaces is consistent with the interface requirements and designs. The Interface Assembly Testing process detailed in this section demonstrates the framework followed by the EAI Team to ensure that the quality of each interface developed is consistent from release to release.

3.1.1 Scope

The scope of Interface Assembly Testing is to test interfaces between the EAI Bus and the Trading Partners. To assist in the validation of the EAI Core functionality for Release 3.0, the EAI Core team developed a test driver application commonly referred to as a test stub. This test driver application does not provide any business functionality, but simulates the Trading Partner actions by providing a user interface for entering or retrieving message data from a file. The test stub then sends messages for processing. In addition, the test stubs helps to isolate problems with the EAI Bus before testing with the trading partners. The various custom components developed by the COD EAI Interfaces team have been tested to ensure that the components interact appropriately with each other.

A COD test stub simulates the responses expected by the Trading Partner from COD for Assembly testing. See the diagram below.



3.1.2 Test Planning

Thorough planning of test cases ensures that end interfaces are robust. Listed below are the steps required to plan test conditions.

1. Define Test Scenarios and Test Cycles
2. Create Test Data
3. Develop Test Environment and Load Data

3.1.3 Define Test Scenarios and Test Cycles

The testing process for the EAI Core Architecture Release 3.0 legacy systems includes the creation of Test Scenarios that validate the functionality for each interface. Each scenario contains three types of test cycles:

- Normal – This cycle validates the interface functionality under normal operating conditions.
- Expected Error – This cycle validates the interface functionality handling of know or expected error conditions.
- Unexpected Error - This cycle validates the interface functionality handling of application or environment failures or outages such as server, network, or database connectivity outages.

For each interface tested, the following sections will be defined for each scenario:

- Test Scenario Description
- Test Scenario Dependencies
- Test Scenario Inputs
- Test Scenario Expected Results

3.1.3.1 Test Scenario Description

This section provides the objective and an overview of the test to be performed. It also outlines functions exercised relative to the MQSeries, MQSeries Integrator and legacy system tests.

3.1.3.2 Test Scenario Dependencies

This section defines the system dependencies, both hardware and software that must be met prior to test execution.

3.1.3.3 Test Scenario Inputs

This section provides a description of the data required to execute the test scenario.

3.1.3.4 Test Scenario Expected Results

This section provides the expected results, or output, of the particular test scenario. The expected results for each test scenario are the same as if the transaction were executed on each system without using MQSeries as the message transport. MQSeries is necessary to transport files via the EAI Bus. However, expected results are independent of the transport mechanism used to move these files, when no data transformation is required. Acceptance of the test is gained by demonstrating to Accenture and FSA that the transaction is executed successfully (i.e. the expected results are returned).

3.1.4 Create Test Data

Realistic test data was created to facilitate the functional, end-to-end test. It is important that test data, once established, remains static so that the Test Team can create expected results that will accurately reflect the environment in which product test is being conducted. The reference and application data that were established for the test environment was backed up to establish the test bed. This test bed was used to refresh the test environment if needed by the test cycles.

3.1.5 Develop Test Environment and Load Data

Testing was executed in a test environment. An exact Production environment replica was created with production-like hardware and software configurations and process and data distributions.

3.1.6 Test Execution

The execution consists of the following steps:

1. Execute Test Scenarios/Cycles
2. Verify Test Results
3. Create System Investigation Requests (SIRs) and Accept Fixes
4. Verify completion of exit/entry requirements

3.1.6.1 Execute Test Conditions/Scenarios

The Test Cycles have been executed for each interface scenario.

3.1.6.2 Verify Test Results

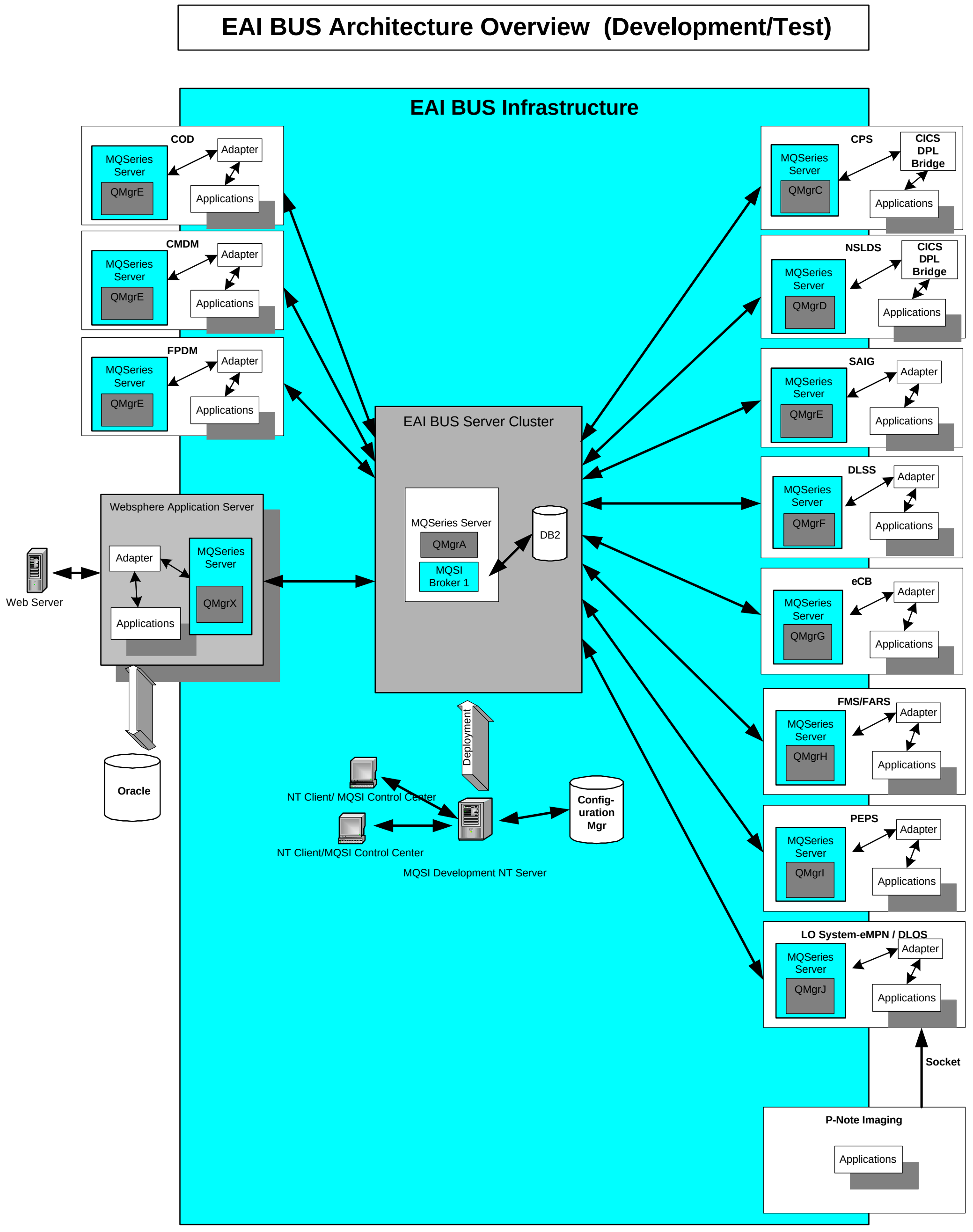
The test results were compared with the expected results in the test script and any discrepancies were noted and the appropriate SIR was created.

3.1.6.3 Create System Investigation Reports (SIRs) and Accept Fixes

Each time a discrepancy between the actual results and the expected results was found, a system investigation request (SIR) was created to formally log the issue. The discrepancy may be, but is not limited to, an error, a defect, an environmental problem, a usability enhancement or a miscalculated expected result.

3.1.7 Test Environment

The intent of the diagram is to show the components of the EAI Bus Architecture implemented for Release 3.0 of the EAI Core. The location of MQSeries, MQSI, databases and adapters are shown.



.

4 EAI COMPONENT TESTS

4.1 FARS Retirement DLSS – CMDM Interface - Batch

The Credit Management Data Mart (CMDM) receives daily feeds of financial transaction information from the FSA Financial Management System (FMS) system. These financial transactions provide the CMDM reporting capability for the Direct Loan portfolio of FSA. The financial transactions originate in the Direct Loan Servicing System (DLSS) and are posted (in summary form) within the FMS system. The supporting detail is then extracted from FMS and delivered to the CMDM via the EAI bus architecture.

4.1.1 DLSS to CMDM – Batch Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for the DLSS to CMDM interface is based on a bulk file transfer from DLSS to CMDM system. The EAI development team provided parameters to the Open VMS (Virtual Memory System) step that initiates the file transfer. Data Integrator is configured to manage the file transfers and initiate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.1.2 DLSS to CMDM – Batch Test Scenario Dependencies

To execute the DLSS to FPDM interface test scenario the following dependencies were met:

- MQSeries Messaging and queue managers on each of the following systems are operational, logically referred to as CRDEV2, SU35E5 and SU35E16 or SU35E17.
- Data Integrator must be operational on either of the following systems, logically referred to as CRDEV2, SU35E5 and SU35E16 or SU35E17.
- Data Integrator has access to read and write files on DLSS & CMDM systems.
- The Data Integrator send and post processing shell script that will transmit the file has been created and is operational.

4.1.3 DLSS to CMDM – Batch Test Scenario Inputs

This interface transports file from DLSS to the EAI Bus and finally to CMDM via Data Integrator. The EAI Bus is simply a pass through for the files and, no transformation is performed. Two types of test data were used in the Assembly Test:

1. A test file was created by the EAI team to test the connectivity between the DLSS, EAI Bus and CMDM.
2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the DLSS to CMDM interface.

4.1.4 DLSS to CMDM – Batch Test Scenario Expected Results

The EAI Core team provided ACS (DLSS System Administrators) with the commands necessary to initiate file transfers from the DLSS test system to the CMDM test system. All batch file transfers were initiated at DLSS by ACS. The files were written to a target directory

(/ftparea/cm_ftp_in/SrcFiles/Demographic) on CMDM and verified by the FARS Retirement team.

4.2 FARS Retirement FMS – CMDM Interface - Batch

This section is related to section 4.1. Please see section 4.1 for additional details.

4.2.1 FMS to CMDM – Batch Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for the FMS to CMDM interface is based on a bulk file transfer from DLSS to CMDM system. The EAI development team provided a shell script that initiates the file transfer. Data Integrator was configured to manage the file transfers and initiate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.2.2 FMS to CMDM – Batch Test Scenario Dependencies

To execute the FMS to CMDM interface test scenario the following dependencies were met:

- MQSeries Messaging and queue managers on each of the following systems are operational, logically referred to as HPL6, SU35E5 and SU35E16 or SU35E17.
- Data Integrator must be operational on either of the following systems, logically referred to as HPL6, SU35E5 and SU35E16 or SU35E17.
- Data Integrator has access to read and write files on FMS & CMDM systems.
- The Data Integrator send and post processing shell script for transmitting files has been created and is operational.

4.2.3 FMS to CMDM – Batch Test Scenario Inputs

This interface transports files via Data Integrator and MQSeries, it flows from FMS to the EAI Bus, to CMDM. The EAI Bus is simply a pass through for the files and, no transformation is performed. Two types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the FMS, EAI Bus, and CMDM.
2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the FMS to CMDM interface.

4.2.4 FMS to CMDM – Batch Test Scenario Expected Results

The EAI Core team provided the FARS Retirement team with a script to initiate file transfers from the FMS test system to the CMDM test system. The script is capable of detecting and transferring multiple files of each file type. The FARS Retirement team verified that the files were successfully transferred.

4.3 Financial Partner Data Mart PEPS – FPDM Interface - Batch

The Financial Partner Data Mart (FPDM) feeds financial data from PEPS system to FPDM (Informatica). The EAI batch enablement of the FPDM interface provides the capability to send and receive flat files to and from PEPS to FPDM.

4.3.1 PEPS to FPDM – Batch Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for the PEPS to FPDM interface is based on a bulk file transfer from PEPS to FPDM system. The EAI development team provided a shell script to initiate the file transfer. Data Integrator has been configured to manage the file transfers and initiate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.3.2 PEPS to FPDM – Batch Test Scenario Dependencies

To execute the PEPS to FPDM interface test scenario the following dependencies were met:

- MQSeries Messaging and queue managers on each of the following systems are operational, logically referred to as HPK1, SU35E5 and SU35E16 or SU35E17.
- Data Integrator must be operational on either of the following systems, logically referred to as HPK1, SU35E5 and SU35E16 or SU35E17.
- Data Integrator has access to read and write files on PEPS & FPDM systems.
- The Data Integrator send and post processing shell script that will transmit the file has been created and is operational.

4.3.3 PEPS to FPDM – Batch Test Scenario Inputs

The PEPS to FPDM interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries, it flows from PEPS to the EAI Bus, to FPDM. The EAI Bus is simply a pass through for the files and no transformation is performed. Two types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the PEPS, EAI Bus, and FPDM.
2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the PEPS to FPDM interface.

Refer to Appendix B for the detail test conditions.

4.3.4 PEPS to FPDM – Batch Test Scenario Expected Results

The Release 3.0 EAI Core team executed this PEPS to FPDM Interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix B for the expected results from the test conditions.

4.4 Financial Partner Data Mart NSLDS – FPDM Interface - Batch

The Financial partner Data Mart (FPDM) feeds financial data from the NSLDS system to FPDM (Informatica). The EAI batch enablement of the FPDM interface provides the capability to send and receive flat files to and from NSLDS to FPDM.

4.4.1 NSLDS to FPDM – Batch Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for the NSLDS to FPDM interface is based on a bulk file transfer from NSLDS to FPDM system. The EAI development team provided a shell script that initiates the file transfer. Data Integrator has been configured to manage the file transfers and imitate the start of the application process at different stages of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.4.2 NSLDS to FPDM – Batch Test Scenario Dependencies

To execute the NSLDS to FPDM interface test scenario the following dependencies were met:

- MQSeries Messaging and queue managers on each of the following systems are operational, logically referred to as NSLDS (Test), SU35E5 and SU35E16 or SU35E17.
- Data Integrator must be operational on either of the following systems, logically referred to as NSLDS (Test), SU35E5 and SU35E16 or SU35E17.
- Data Integrator has access to read and write files on NSLDS & FPDM systems.
- Data Integrator parameters have been provided for to complete the Job Control Language (JCL) on NSLDS to trigger the send process.
- Data Integrator post processing shell script that will transmit the file has been created and is operational.

4.4.3 NSLDS to FPDM – Batch Test Scenario Inputs

The NSLDS to FPDM interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries, it flows from NSLDS to the EAI Bus, to FPDM. The EAI Bus is simply a pass through for the files and, no transformation is performed. Two types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the NSLDS, EAI Bus, and FPDM.
2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the NSLDS to FPDM interface.

Refer to Appendix B for the detail test conditions.

4.4.4 NSLDS to FPDM – Batch Test Scenario Expected Results

The Release 3.0 EAI Core team executed this PEPS to FPDM interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix B for the expected results from the test conditions.

4.5 eCampus Based PEPS - eCB Interface - Batch

The eCampus Based (eCB) system receives school data from the PEPS system. The EAI batch enablement of the eCB interface provides the capability to send and receive flat files from PEPS to eCB.

4.5.1 PEPS to eCB – Batch Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for the PEPS to eCB interface is based on a bulk file transfer from PEPS to the eCB system. The EAI development team provided a shell script that initiates the file transfer. Data Integrator has been configured to manage the file transfers and initiate the start of the application process at different stages of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.5.2 PEPS to eCB – Batch Test Scenario Dependencies

To execute the PEPS to eCB interface test scenario the following dependencies were met:

- MQSeries Messaging and queue managers on each of the following systems are operational, logically referred to as HPK1, SU35E5 and SU35E16 or SU35E17.
- Data Integrator must be operational on either of the following systems, logically referred to as HPK1, SU35E5 and SU35E16 or SU35E17.
- Data Integrator has access to read and write files on PEPS & eCB systems.
- The Data Integrator send and post processing shell script that will transmit the file has been created and is operational.

4.5.3 PEPS to eCB – Batch Test Scenario Inputs

This interface transports files via Data Integrator and MQSeries, it flows from PEPS to the EAI Bus, to eCB and eCB to the EAI Bus, to PEPS. The EAI Bus is simply a pass through for the files and, no transformation is performed. Four types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the PEPS, EAI Bus, and eCB.
2. A test file was created by the EAI team to test the connectivity between the eCB, EAI Bus, and PEPS.
3. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the PEPS to eCB interface.

Refer to Appendix C for the detail test conditions.

4.5.4 PEPS to eCB Interface – Batch Test Scenario Expected Results

The Release 3.0 EAI Core team executed this PEPS to eCB interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix C for the expected results from the test conditions.

4.6 eCampus Based eCB - FMS Interface - Batch

The eCampus Based (eCB) system sends financial data to the FMS system. The EAI batch enablement of the eCB interface provides the capability to send and receive flat files from eCB to FMS.

4.6.1 eCB to FMS – Batch Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for the eCB to FMS interface is based on a bulk file transfer from eCB to the FMS system. The EAI development team provided a shell script that initiates the file transfer. Data Integrator has been configured to manage the file transfers and initiate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.6.2 eCB to FMS – Batch Test Scenario Dependencies

To execute the eCB to FMS interface test scenario the following dependencies were met:

- MQSeries Messaging and queue managers on each of the following systems are operational, logically referred to as HPL6, SU35E5 and SU35E16 or SU35E17.
- Data Integrator must be operational on either of the following systems, logically referred to as HPL6, SU35E5 and SU35E16 or SU35E17.
- Data Integrator has access to read and write files on eCB & FMS systems.
- The Data Integrator send and post processing shell script that will transmit file has been created and is operational.

4.6.3 eCB to FMS – Batch Test Scenario Inputs

The test files were created by the eCB application team and are of the same size as the eCB production files. The EAI team executed the following test conditions:

- Send files from eCB to EAI Bus
- Send files from EAI Bus to FMS
- Send files from FMS to EAI Bus (flat file transfer only – no MQSI message flow)
- Send files from EAI Bus to eCB (flat file transfer only – no MQSI message flow)

Refer to Appendix C for the detail test conditions.

4.6.4 eCB to FMS – Batch Test Scenario Expected Results

The Release 3.0 EAI Core team executed this eCB to FMS interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix C for the expected results from the test conditions.

4.7 COD Interfaces

4.7.1 COD to CPS Interface

4.7.1.1 Abbreviated Applicant File

The Abbreviated Applicant data is transmitted from CPS to COD whenever CPS processes applications. The needed data is similar to the file currently provided to RFMS with a few additional fields. In addition, the file will contain not only all Pell eligible applicants, but also all applicants with an Expected Family Contribution greater than or equal to zero.

4.7.1.1.1 Abbreviated Applicant File - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure Abbreviated Applicant File interface is based on a bulk file transfer from CPS to COD system. The EAI development team provided parameters to Job Control Language (JCL) step that initiates the file transfer. Data Integrator has been configured to manage the file transfers and imitate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.7.1.1.2 Abbreviated Applicant File – Test Scenario Dependencies

To transfer the Abbreviated Applicant File interface test scenario the following dependencies were met:

- CPS must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- CPS must have MQSeries and Data Integrator running.
- CPS must produce the Abbreviated Applicant File and place it into an agreed upon directory for processing.
- EAI Bus must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- EAI Bus must have MQSeries and Data Integrator running.
- CPS must make the agreed upon directory read/write-able for the Utility.
- CPS must provide a compiler for C++ for use by the EAI Team.
- COD must have IBM MQSeries v2.1 and Commerce Quest Data Integrator v4.0.1 installed.
- COD must have MQSeries and Data Integrator running.
- COD must have MQSeries running.
- COD must place components necessary to retrieve the Abbreviated Applicant File from the appropriate MQSeries Queue.

4.7.1.1.3 Abbreviated Applicant File - Test Scenario Inputs

The Abbreviated Applicant File interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries, it flows from CPS to the EAI Bus, to COD. The EAI Bus is simply a pass through for the files and no transformation is performed. Two types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the CPS, EAI Bus, and COD.
2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the CPS to COD interface.

Refer to Appendix D for the detail test conditions.

4.7.1.2 Abbreviated Applicant File – Test Scenario Expected Results

The Release 3.0 EAI Core team executed this Abbreviated Applicant File interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix D for the expected results from the test conditions.

4.7.1.3 Pell Institution Universe File

Today, RFMS produces the Pell Institution Universe File to provide CPS with a weekly record of Pell Institution processing. This file is extracted and transmitted to CPS upon successful completion of the RFMS weekly processing cycle. The file contains all institution data from RFMS for Pell IDs beginning with a 0 and will be sorted in Pell ID order.

COD updates Institution data for both Pell and Direct Loan school interactions. This updated data for Pell institutions is provided to CPS on a daily basis.

4.7.1.3.1 Pell Institution Universe File - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for Pell Institution Universe File interface is based on a bulk file transfer from COD to CPS system. The EAI development team provided parameters to Job Control Language (JCL) that initiates the file transfer. Data Integrator has been configured to manage the file transfers and imitate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.7.1.3.2 Pell Institution Universe File – Test Scenario Dependencies

To transfer the Pell Institution Universe File interface test scenario the following dependencies were met:

- COD must have IBM MQSeries v2.1 and Commerce Quest Data Integrator v4.0.1 installed.
- COD must have MQSeries and Data Integrator running.
- COD must place components necessary to create the Pell Institution Universe File onto the appropriate MQSeries Queue.

- COD must send an MQSeries Message that will trigger the utility on the CPS system when the entire batch has been committed to MQSeries queues on the EAI Bus.
- EAI Bus must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- EAI Bus must have MQSeries and Data Integrator running.
- CPS must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- CPS must have MQSeries and Data Integrator running.
- CPS must accept the Pell Institution Universe File into an agreed upon directory for processing.
- CPS must make the agreed upon directory read/write-able for the Utility.
- CPS must provide a compiler for C++ for use by the EAI Team.

4.7.1.3.3 Pell Institution Universe File - Test Scenario Inputs

The Pell Institution Universe File interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries, it flows from COD to the EAI Bus, to CPS. The EAI Bus is simply a pass through for the files and, no transformation is performed. Two types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the COD, EAI Bus, and CPS.
2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the COD to CPS interface.

Refer to Appendix D for the detail test conditions.

4.7.1.3.4 Pell Institution Universe File – Test Scenario Expected Results

The Release 3.0 EAI Core team executed this Pell Institution Universe File test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix D for the expected results from the test conditions.

4.7.1.4 Pell Recipient File

Pell Recipient data is produced for several other organizations internal and external to FSA. CPS uses the data for verification analysis and for end-of-year reporting. CPS will need the first file in April 2003 with data for the 02-03 Award Year. CPS needs the file several other times throughout the year, with the final delivery in December 2003 for the 02-03 Award Year. Due to the size of this data, it is currently provided to the other systems via tape. COD and CPS will work towards providing electronic delivery of this file. CPS would like to have an electronic mechanism for requesting the file.

4.7.1.4.1 Pell Recipient File - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for Pell Recipient File interface is based on a bulk file transfer from COD to CPS system. The EAI development team provided parameters to Job Control Language (JCL) that initiates the file transfer. Data Integrator has been configured to manage the file transfers and imitate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.7.1.4.2 Pell Recipient File – Test Scenario Dependencies

To transfer the Pell Recipient File interface test scenario the following dependencies were met:

- COD must have IBM MQSeries v2.1 and Commerce Quest Data Integrator v4.0.1 installed.
- COD must have MQSeries and Data Integrator running.
- COD must place components necessary to create the Pell Recipient File onto the appropriate MQSeries Queue.
- COD must send an MQSeries Message that will trigger the utility on the CPS system when the entire batch has been committed to MQSeries queues on the EAI Bus.
- EAI Bus must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- EAI Bus must have MQSeries and Data Integrator running.
- CPS must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- CPS must have MQSeries and Data Integrator running.
- CPS must accept the Pell Recipient File into an agreed upon directory for processing.
- CPS must make the agreed upon directory read/write-able for the Utility.
- CPS must provide a compiler for C++ for use by the EAI Team.

4.7.1.4.3 Pell Recipient File - Test Scenario Inputs

The Pell Recipient File interface test scenario included three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries, it flows from COD to the EAI Bus, to CPS. The EAI Bus is simply a pass through for the files and, no transformation is performed. Two types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the COD, EAI Bus, and CPS.
2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the COD to CPS interface.

Refer to Appendix D for the detail test conditions.

4.7.1.4.4 Pell Recipient File – Test Scenario Expected Results

The Release 3.0 EAI Core team executed this Pell Recipient File interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix D for the expected results from the test conditions.

4.7.2 COD to DLOS Interface

4.7.2.1 PLUS MPN Record

There are two times when a PLUS MPN Record is generated: 1) after processing an image on the Imaging system, and 2) after a user files a PLUS MPN on-line. A PLUS MPN record is generated and sent to COD by the eMPN component on the LOWEB Server upon the successful completion of the PLUS eMPN process. This interface is transactional and is used as needed in real time via MQ Series.

The Imaging system will generate and send to COD a PLUS MPN record upon the successful completion of the imaging process for each paper PLUS MPN. This interface consists of a two-step process of sending the data to the LOWEB Server via a socket connection, then using MQSeries to send the message to COD. The transmission from the Imaging Server to the LOWEB Server is entirely the responsibility of EDS.

4.7.2.1.1 PLUS MPN Record - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for PLUS MPN Record interface is based on a real time message transfer from LOWEB to COD system. The EAI team developed custom Java adapter code for LOWEB to place messages on the MQSeries queue. MQSeries queues and channels enable messages to be transferred in real time.

4.7.2.1.2 PLUS MPN Record - Test Scenario Dependencies

To execute the PLUS MPN Record interface test scenario the following dependencies were met:

- LOWEB must have IBM MQSeries v5.2 installed.
- LOWEB must have MQSeries running.
- LOWEB must produce the PLUS MPN Record and place it on the appropriate MQSeries Queue.
- LOWEB must have Java Runtime Environment and Java adapter code.
- EAI Bus must have IBM MQSeries v5.2 installed.
- EAI Bus must have MQSeries running.
- COD must have IBM MQSeries v2.1 installed.
- COD must have MQSeries running.
- COD must place components necessary to retrieve the PLUS MPN Record from the appropriate MQSeries Queue.

4.7.2.1.3 PLUS MPN Record - Test Scenario Inputs

The PLUS MPN Record interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports data via MQSeries; it flows from LOWEB to the EAI Bus to COD. The EAI Bus is simply a pass through for the messages and no transformation is performed. One type of test data was used in the system integrated test phase:

1. A test PLUS MPN Record was created by the EAI team to test the connectivity between the LOWEB, EAI Bus, and COD.

Refer to Appendix E for the detail test conditions.

4.7.2.1.4 PLUS MPN Record - Test Scenario Expected Results

The Release 3.0 EAI Core team executed this PLUS MPN Record interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix E for the expected results from the test conditions.

4.7.2.2 PLUS Endorser Record

The Imaging system generates and sends to COD the PLUS Endorser record upon the successful completion of the imaging process for each endorser addendum. This interface consists of a two-step process of sending the data to the LOWEB Server via a socket connection, then using MQSeries to send the message to COD. The transmission from the Imaging Server to the LOWEB Server is entirely the responsibility of EDS.

4.7.2.3 PLUS Endorser Record - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for PLUS Endorser Record interface was based on a real time message transfer from LOWEB to COD system. The EAI team developed custom Java adapter code for LOWEB to place messages on the MQSeries queue. MQSeries queues and channels enable messages to be transferred in real time.

4.7.2.3.1 PLUS Endorser Record - Test Scenario Dependencies

To execute the PLUS Endorser Record interface test scenario the following dependencies were met:

- LOWEB must have IBM MQSeries v5.2 installed.
- LOWEB must have MQSeries running.
- LOWEB must produce the PLUS Endorser Record and place it on the appropriate MQSeries Queue.
- LOWEB must have Java Runtime Environment and Java adapter code.
- EAI Bus must have IBM MQSeries v5.2 installed.
- EAI Bus must have MQSeries running.
- COD must have IBM MQSeries v2.1 installed.
- COD must have MQSeries running.
- COD must place components necessary to retrieve the PLUS MPN Record from the appropriate MQSeries Queue.

4.7.2.3.2 PLUS Endorser Record - Test Scenario Inputs

The PLUS Endorser Record interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports data via MQSeries; it flows from LOWEB to the EAI Bus to COD. The EAI Bus is simply a pass through for the messages and no transformation is performed. One type of test data was used in the system integrated test phase:

1. A test PLUS Endorser Record was created by the EAI team to test the connectivity between the LOWEB, EAI Bus, and COD.

Refer to Appendix E for the detail test conditions.

4.7.2.3.3 PLUS Endorser Record - Test Scenario Expected Results

The Release 3.0 EAI Core team executed this PLUS Endorser Record interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix E for the expected results from the test conditions.

4.7.2.3.4 Borrower Validation & MPN ID Number Request/Response

Schools have the option to require that potential borrowers have an award originated in COD before allowing borrowers to complete a MPN. This interface allows the eMPN system to query COD to see if the award has been established by the school for the borrower. The interface is transactional and is used as needed in real time via MQ Series. The response includes the MPN ID if one is available.

4.7.2.3.5 Borrower Validation & MPN ID Number Request - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for Borrower Validation & MPN ID Number Request interface is based on a real time message transfer from LOWEB to COD system. The EAI team developed custom Java adapter code for LOWEB to place messages on the MQSeries queue. MQSeries queues and channels enable messages to be transferred in real time.

4.7.2.3.6 Borrower Validation & MPN ID Number Request - Test Scenario Dependencies

To execute the Borrower Validation & MPN ID Number Request interface test scenario the following dependencies must be met:

- LOWEB must have IBM MQSeries v5.2 installed.
- LOWEB must have MQSeries running.
- LOWEB must produce the Borrower Validation & MPN ID Number Request and place it on the appropriate MQSeries Queue.
- LOWEB must have Java Runtime Environment and Java adapter code.
- EAI Bus must have IBM MQSeries v5.2 installed.
- EAI Bus must have MQSeries running.
- COD must have IBM MQSeries v2.1 installed.
- COD must have MQSeries running.
- COD must place components necessary to retrieve the Borrower Validation & MPN ID Number Request from the appropriate MQSeries Queue.

4.7.2.3.7 Borrower Validation & MPN ID Number Request - Test Scenario Inputs

The Borrower Validation & MPN ID Number Request interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports data via

MQSeries; it flows from LOWEB to the EAI Bus to COD. The EAI Bus is simply a pass through for the messages and no transformation is performed. One type of test data was used in the system integrated test phase:

1. A test Borrower Validation & MPN ID Number Request was created by the EAI team to test the connectivity between the LOWEB, EAI Bus, and COD.

Refer to Appendix E for the detail test conditions.

4.7.2.3.8 Borrower Validation & MPN ID Number Request - Test Scenario Expected Results

The Release 3.0 EAI Core team executed this Borrower Validation & MPN ID Number Request test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix E for the expected results from the test conditions.

4.7.2.3.9 Borrower Validation & MPN ID Number Response - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for Borrower Validation & MPN ID Number Response interface is based on a real time message transfer from COD to LOWEB system. The EAI team developed custom Java adapter code for LOWEB to retrieve messages on the MQSeries queue. MQSeries queues and channels enable messages to be transferred in real time.

4.7.2.3.10 Borrower Validation & MPN ID Number Response - Test Scenario Dependencies

To execute the Borrower Validation & MPN ID Number Response interface test scenario the following dependencies were met:

- COD must have IBM MQSeries v2.1 installed.
- COD must have MQSeries running.
- COD must place components necessary to retrieve the Borrower Validation & MPN ID Number Response from the appropriate MQSeries Queue.
- EAI Bus must have IBM MQSeries v5.2 installed.
- EAI Bus must have MQSeries running.
- LOWEB must have IBM MQSeries v5.2 installed.
- LOWEB must have MQSeries running.
- LOWEB must produce the Borrower Validation & MPN ID Number Response and place it on the appropriate MQSeries Queue.
- LOWEB must have Java Runtime Environment and Java adapter code.

4.7.2.3.11 Borrower Validation & MPN ID Number Response - Test Scenario Inputs

The Borrower Validation & MPN ID Number Response interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports data via MQSeries; it flows from COD to the EAI Bus to LOWEB. The EAI Bus is simply a pass through for the messages and no transformation is performed. One type of test data was used in the system integrated test phase:

1. A test Borrower Validation & MPN ID Number Response was created by the EAI team to test the connectivity between the COD, EAI Bus, and LOWEB.

Refer to Appendix E for the detail test conditions.

4.7.2.3.12 Borrower Validation & MPN ID Number Response - Test Scenario Expected Results

The Release 3.0 EAI Core team executed this Borrower Validation & MPN ID Number Response interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix E for the expected results from the test conditions.

4.7.2.4 LO Web and DLOS Credit Checks

The LO Web application and DLOS Batch application shares an interface on the LO Web server to retrieve credit check information from COD. The LO Web application offers school users the ability to perform credit checks for potential PLUS loan borrowers and DLOS will need the ability to obtain credit check information for prior year (before AY 02-03) PLUS loans. This interface is transactional and is used as needed in real time via MQ Series.

4.7.2.4.1 LO Web and DLOS Credit Checks Request - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for LO Web and DLOS Credit Checks Request interface is based on a real time message transfer from LOWEB to COD system. The EAI team developed custom Java adapter code for LOWEB to place messages on the MQSeries queue. MQSeries queues and channels enable messages to be transferred in real time.

4.7.2.4.2 LO Web and DLOS Credit Checks Request - Test Scenario Dependencies

To execute the LO Web and DLOS Credit Checks Request interface test scenario the following dependencies were met:

- LOWEB must have IBM MQSeries v5.2 installed.
- LOWEB must have MQSeries running.
- LOWEB must produce the LO Web and DLOS Credit Checks Request and place it on the appropriate MQSeries Queue.
- LOWEB must have Java Runtime Environment and Java adapter code.
- EAI Bus must have IBM MQSeries v5.2 installed.
- EAI Bus must have MQSeries running.
- COD must have IBM MQSeries v2.1 installed.
- COD must have MQSeries running.
- COD must place components necessary to retrieve the LO Web and DLOS Credit Checks Request from the appropriate MQSeries Queue.

4.7.2.4.3 LO Web and DLOS Credit Checks Request - Test Scenario Inputs

The LO Web and DLOS Credit Checks Request interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports data via MQSeries; it flows from LOWEB to the EAI Bus to COD. The EAI Bus is simply a pass through for the messages and no transformation is performed. One type of test data was used in the system integrated test phase:

1. A test LO Web and DLOS Credit Checks Request was created by the EAI team to test the connectivity between the LOWEB, EAI Bus, and COD.

Refer to Appendix E for the detail test conditions.

4.7.2.4.4 LO Web and DLOS Credit Checks Request - Test Scenario Expected Results

The Release 3.0 EAI Core team executed this LO Web and DLOS Credit Checks Request interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix E for the expected results from the test conditions.

4.7.2.4.5 LO Web and DLOS Credit Checks Response - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for LO Web and DLOS Credit Checks Response interface is based on a real time message transfer from COD to LOWEB system. The EAI team developed custom Java adapter code for LOWEB to retrieve messages on the MQSeries queue. MQSeries queues and channels enable messages to be transferred in real time.

4.7.2.4.6 LO Web and DLOS Credit Checks Response - Test Scenario Dependencies

To execute the LO Web and DLOS Credit Checks Response interface test scenario the following dependencies were met:

- COD must have IBM MQSeries v2.1 installed.
- COD must have MQSeries running.
- COD must place components necessary to retrieve the LO Web and DLOS Credit Checks Response from the appropriate MQSeries Queue.
- EAI Bus must have IBM MQSeries v5.2 installed.
- EAI Bus must have MQSeries running.
- LOWEB must have IBM MQSeries v5.2 installed.
- LOWEB must have MQSeries running.
- LOWEB must produce the LO Web and DLOS Credit Checks Response and place it on the appropriate MQSeries Queue.
- LOWEB must have Java Runtime Environment and Java adapter code.

4.7.2.4.7 LO Web and DLOS Credit Checks Response - Test Scenario Inputs

The LO Web and DLOS Credit Checks Response interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports data via

MQSeries; it flows from COD to the EAI Bus to LOWEB. The EAI Bus is simply a pass through for the messages and no transformation is performed. One type of test data was used in the system integrated test phase:

1. A test LO Web and DLOS Credit Checks Response was created by the EAI team to test the connectivity between the COD, EAI Bus, and LOWEB.

Refer to Appendix E for the detail test conditions.

4.7.2.4.8 LO Web and DLOS Credit Checks Response - Test Scenario Expected Results

The Release 3.0 EAI Core team executed this LO Web and DLOS Credit Checks Response interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix E for the expected results from the test conditions.

4.7.2.5 P-Note Linking Request

This interface allows COD to request promissory note information for Stafford subsidized and unsubsidized loans from DLOS to verify that Direct Loan awards in COD have a corresponding promissory note. This is a batch interface between COD and DLOS and is transmitted directly between the two systems daily. This interface is a direct connection from COD to DLOS. COD is responsible for this interface.

4.7.2.5.1 Date Change Payment Trigger

When COD generates the first disbursement for direct loan, the actual disbursement date is sent via this interface so that DLOS can recalculate the expiration date for the promissory note. This interface is a direct connection from COD to DLOS. COD is responsible for this interface.

4.7.2.5.2 Unsolicited MPN and Link Response

The response to the P-Note link request includes the identifier for the requested loan, if found, or the status as pending. In addition, data about promissory notes that have been received without a corresponding award request are provided via this interface. This interface is a direct connection from DLOS to COD. COD is responsible for this interface.

4.7.2.5.3 MPN Status Change

For AY 02-03, DLOS will maintain Stafford subsidized and unsubsidized P-Note information. This interface allows DLOS to notify COD of any changes to promissory note information for subsidized or unsubsidized loans. This is a batch interface and is transmitted directly from DLOS to COD daily. This interface is a direct connection from DLOS to COD. COD is responsible for this interface.

4.7.3 COD to DLSS Overview

4.7.3.1 DLSS Batch Feed

This feed contains daily updates of Loan Bookings, School Data, and other DLSS-relevant data that is provided by COD to DLSS.

4.7.3.1.1 DLSS Batch Feed - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for DLSS Batch Feed interface is based on a bulk file transfer from COD to DLSS system. The EAI development team provided parameters to Job Control Language (JCL) that initiates the file transfer. Data Integrator has been configured to manage the file transfers and imitate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.7.3.1.2 DLSS Batch Feed – Test Scenario Dependencies

To transfer the DLSS Batch Feed interface test scenario the following dependencies were met:

- COD must have IBM MQSeries v2.1 and Commerce Quest Data Integrator v4.0.1 installed.
- COD must have MQSeries and Data Integrator running.
- COD must place components necessary to create the DLSS Batch Feed file onto the appropriate MQSeries Queue.
- COD must send an MQSeries Message that will trigger the utility on the DLSS system when the entire batch has been committed to MQSeries queues on the EAI Bus.
- EAI Bus must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- EAI Bus must have MQSeries and Data Integrator running.
- DLSS must have IBM MQSeries v2.2.1.1 and Commerce Quest Data Integrator v4.0.1 installed.
- DLSS must have MQSeries and Data Integrator running.
- DLSS must accept the DLSS Batch Feed into an agreed upon directory for processing.
- DLSS must make the agreed upon directory read/write-able for the Utility.
- DLSS must provide a compiler for C++, v6.2 on an ALPHA 4100, with VMS7.2-1 for use by the EAI Team.

4.7.3.1.3 DLSS Batch Feed - Test Scenario Inputs

The DLSS Batch Feed interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries; it flows from COD to the EAI Bus to DLSS. The EAI Bus is simply a pass through for the files and no transformation is performed. Two types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the COD, EAI Bus, and DLSS.
2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the COD to DLSS interface.

Refer to Appendix F for the detail test conditions.

4.7.3.1.4 DLSS Batch Feed – Test Scenario Expected Results

The Release 3.0 EAI Core team executed this DLSS Batch Feed interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix F for the expected results from the test conditions.

4.7.3.2 DLSS Batch Response

This feed contains daily responses from DLSS to COD to updates of Loan Bookings, School Data and other DLSS-relevant previously provided by COD to DLSS.

4.7.3.2.1 DLSS Batch Response - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for DLSS Batch Response interface is based on a bulk file transfer from DLSS to COD system. The EAI development team provided parameters to Open VMS (Virtual Memory System) step that initiates the file transfer. Data Integrator has been configured to manage the file transfers and imitate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.7.3.2.2 DLSS Batch Response – Test Scenario Dependencies

To transfer the DLSS Batch Response interface test scenario the following dependencies were met:

- DLSS must have IBM MQSeries v2.2.1.1 and Commerce Quest Data Integrator v4.0.1 installed.
- DLSS must have MQSeries and Data Integrator running.
- DLSS must produce the DLSS Batch Response file and place it into an agreed upon directory for processing.
- DLSS must make the agreed upon directory read/write-able for the Utility
- DLSS must provide a compiler for C++, v6.2 on an ALPHA 4100, with VMS7.2-1 for use by the EAI Team.
- EAI Bus must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- EAI Bus must have MQSeries and Data Integrator running.
- COD must have IBM MQSeries v2.1 and Commerce Quest Data Integrator v4.0.1 installed.
- COD must have MQSeries and Data Integrator running.

- COD must have MQSeries running.
- COD must place components necessary to retrieve the DLSS Batch Response from the appropriate MQSeries Queue.

4.7.3.2.3 DLSS Batch Response - Test Scenario Inputs

The DLSS Batch Response interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries; it flows from DLSS to the EAI Bus to COD. The EAI Bus is simply a pass through for the files and no transformation is performed. Two types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the DLSS, EAI Bus, and COD.
2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the DLSS to COD interface.

Refer to Appendix F for the detail test conditions.

4.7.3.2.4 DLSS Batch Response – Test Scenario Expected Results

The Release 3.0 EAI Core team executed this DLSS Batch Response interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix F for the expected results from the test conditions.

4.7.3.3 Disbursement Confirmations (RDC's)

This feed contains Confirmations of COD-initiated FLA, FLB, and FLD transactions that took place since the last Disbursement Confirmation File that was sent to COD by DLSS. The process is part of the COD-DLSS Reconciliation Process.

4.7.3.3.1 Disbursement Confirmations - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure Disbursement Confirmation interface is based on a bulk file transfer from DLSS to COD system. The EAI development team provided parameters to Open VMS (Virtual Memory System) step that initiates the file transfer. Data Integrator has been configured to manage the file transfers and imitate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.7.3.3.2 Disbursement Confirmations – Test Scenario Dependencies

To transfer the Disbursement Confirmations interface test scenario the following dependencies were met:

- DLSS must have IBM MQSeries v2.2.1.1 and Commerce Quest Data Integrator v4.0.1 installed.
- DLSS must have MQSeries and Data Integrator running.
- DLSS must produce the Disbursement Confirmations file and place it into an agreed upon directory for processing.

- DLSS must make the agreed upon directory read/write-able for the Utility.
- DLSS must provide a compiler for C++, v6.2 on an ALPHA 4100, with VMS7.2-1 for use by the EAI Team.
- EAI Bus must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- EAI Bus must have MQSeries and Data Integrator running.
- COD must have IBM MQSeries v2.1 and Commerce Quest Data Integrator v4.0.1 installed.
- COD must have MQSeries and Data Integrator running.
- COD must have MQSeries running.
- COD must place components necessary to retrieve the Disbursement Confirmations from the appropriate MQSeries Queue.

4.7.3.3.3 Disbursement Confirmations - Test Scenario Inputs

The Disbursement Confirmations interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries; it flows from DLSS to the EAI Bus to COD. The EAI Bus is simply a pass through for the files and no transformation is performed. Two types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the DLSS, EAI Bus, and COD.
2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the DLSS to COD interface.

Refer to Appendix F for the detail test conditions.

4.7.3.3.4 Disbursement Confirmations – Test Scenario Expected Results

The Release 3.0 EAI Core team executed this Disbursement Confirmations interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix F for the expected results from the test conditions.

4.7.4 COD to FMS Overview

4.7.4.1 Financial Transactions

This interface enables FMS and COD to exchange accounting information. This interface is transactional and is used as needed in real time via MQ Series.

4.7.4.1.1 Financial Transactions Input –Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for the Financial Transactions Input interface is based on a real time message transfer from COD to the FMS system. The EAI team configured MQSeries Integrator (MQSI) and developed custom Java adapter to store messages on the FMS database.

4.7.4.1.2 Financial Transactions Input - Test Scenario Dependencies

To execute the Financial Transactions Input interface test scenario the following dependencies were met:

- COD must have IBM MQSeries v2.1 installed.
- COD must have MQSeries running.
- COD must have the necessary component to place the Financial Transactions Input on the appropriate MQSeries Queue.
- EAI Bus must have IBM MQSeries v5.2 and IBM MQSeries Integrator (MQSI) v2.0.1 installed.
- EAI Bus must have MQSeries and MQSI running.
- FMS must have IBM MQSeries v5.2 and Oracle database installed.
- FMS must have MQSeries and Oracle running.
- FMS must store the Financial Transactions Input in the appropriate Oracle database tables.
- FMS must have Java Runtime Environment and Java adapter code.

4.7.4.1.3 Financial Transactions Input - Test Scenario Inputs

The Financial Transactions Input interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports data via MQSeries; it flows from COD to the EAI Bus to FMS. The EAI Bus passes the message and uses MQSI to convert messages to SQL statements. One type of test data was used in the system integrated test phase:

1. A test Financial Transactions Input was created by the EAI team to test the connectivity between the COD, EAI Bus, and FMS.

Refer to Appendix G for the detail test conditions.

4.7.4.1.4 Financial Transactions Input - Test Scenario Expected Results

The Release 3.0 EAI Core team executed this Financial Transactions Input interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix G for the expected results from the test conditions.

4.7.4.1.5 Financial Transactions Retrieval –Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for the Financial Transactions Retrieval interface is based on a real time message transfer from FMS to the COD system. The EAI team configured MQSeries Integrator (MQSI) and developed custom Java adapter to retrieve messages from the FMS database.

4.7.4.1.6 Financial Transactions Retrieval - Test Scenario Dependencies

To execute the Financial Transactions Retrieval interface test scenario the following dependencies were met:

- FMS must have IBM MQSeries v5.2 and Oracle database installed.
- FMS must have MQSeries and Oracle running.
- FMS must have Financial Transactions stored in the appropriate Oracle database tables.
- FMS must have Java Runtime Environment and Java adapter code.
- EAI Bus must have IBM MQSeries v5.2 and IBM MQSeries Integrator (MQSI) v2.0.1 installed.
- EAI Bus must have MQSeries and MQSI running.
- COD must have IBM MQSeries v2.1 installed.
- COD must have MQSeries running.
- COD must have the necessary component to retrieve the Financial Transactions from the appropriate MQSeries Queue.

4.7.4.1.7 Financial Transactions Retrieval - Test Scenario Inputs

The Financial Transactions Retrieval interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports data via MQSeries; it flows from FMS to the EAI Bus to COD. The EAI Bus passes the message and uses MQSI to convert SQL statements to COD messages. One type of test data was used in the system integrated test phase:

1. A test Financial Transactions Retrieval was created by the EAI team to test the connectivity between the FMS, EAI Bus, and FMS.

Refer to Appendix G for the detail test conditions.

4.7.4.1.8 Financial Transactions Retrieval - Test Scenario Expected Results

The Release 3.0 EAI Core team executed this Financial Transactions Retrieval interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix G for the expected results from the test conditions.

4.7.4.2 School Information

This interface enables COD to send Institution Data to FMS. This interface is transactional and is used as needed in real time via MQ Series.

4.7.4.2.1 School Information Input –Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for the School Information Input interface is based on a real time message transfer from COD to the FMS system. The EAI team configured MQSeries Integrator (MQSI) and developed custom Java adapter to store messages on the FMS database.

4.7.4.2.2 School Information Input - Test Scenario Dependencies

To execute the School Information Input interface test scenario the following dependencies were met:

- COD must have IBM MQSeries v2.1 installed.
- COD must have MQSeries running.
- COD must have the necessary component to place the School Information Input on the appropriate MQSeries Queue.
- EAI Bus must have IBM MQSeries v5.2 and IBM MQSeries Integrator (MQSI) v2.0.1 installed.
- EAI Bus must have MQSeries and MQSI running.
- FMS must have IBM MQSeries v5.2 and Oracle database installed.
- FMS must have MQSeries and Oracle running.
- FMS must store the School Information Input in the appropriate Oracle database tables.
- FMS must have Java Runtime Environment and Java adapter code.

4.7.4.2.3 School Information Input - Test Scenario Inputs

The School Information Input interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports data via MQSeries; it flows from COD to the EAI Bus to FMS. The EAI Bus passes the message and uses MQSI to convert messages to SQL statements. One type of test data was used in the system integrated test phase:

1. A test School Information Input was created by the EAI team to test the connectivity between the COD, EAI Bus, and FMS.

Refer to Appendix G for the detail test conditions.

4.7.4.2.4 School Information Input - Test Scenario Expected Results

The Release 3.0 EAI Core team executed this School Information Input interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix G for the expected results from the test conditions.

4.7.4.2.5 School Information Retrieval –Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for the School Information Retrieval interface is based on a real time message transfer from FMS to the COD system. The EAI team configured MQSeries Integrator (MQSI) and developed custom Java adapter to retrieve messages from the FMS database.

4.7.4.2.6 School Information Retrieval - Test Scenario Dependencies

To execute the School Information Retrieval interface test scenario the following dependencies were met:

- FMS must have IBM MQSeries v5.2 and Oracle database installed.

- FMS must have MQSeries and Oracle running.
- FMS must have School Information stored in the appropriate Oracle database tables.
- FMS must have Java Runtime Environment and Java adapter code.
- EAI Bus must have IBM MQSeries v5.2 and IBM MQSeries Integrator (MQSI) v2.0.1 installed.
- EAI Bus must have MQSeries and MQSI running.
- COD must have IBM MQSeries v2.1 installed.
- COD must have MQSeries running.
- COD must have the necessary component to retrieve the School Information from the appropriate MQSeries Queue.

4.7.4.2.7 School Information Retrieval - Test Scenario Inputs

The School Information Retrieval interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports data via MQSeries; it flows from FMS to the EAI Bus to COD. The EAI Bus passes the message and uses MQSI to convert SQL statements to COD messages. One type of test data was used in the system integrated test phase:

1. A test School Information Retrieval was created by the EAI team to test the connectivity between the FMS, EAI Bus, and FMS.

Refer to Appendix G for the detail test conditions.

4.7.4.2.8 School Information Retrieval - Test Scenario Expected Results

The Release 3.0 EAI Core team executed this School Information Retrieval interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix G for the expected results from the test conditions.

4.7.4.3 Reconciliation and Balancing

This interface is a direct connection with FMS and COD. COD and FMS are responsible for this interface.

4.7.5 COD to NSLDS Overview

4.7.5.1 Pell Recipient Information

The Pell Recipient data transmits from COD to NSLDS daily. The file reports all student data for disbursement transactions processed by COD since the previous transmission. COD produces the Student Disbursement and Eligibility File to provide NSLDS with a daily record of Pell student disbursement processing and eligibility. This file will be extracted and transmitted to NSLDS daily. The file reports all student data for disbursement transactions processed by COD since the previous transmission. NSLDS prepares and returns to COD an error file after receiving and processing the Student Disbursement and Eligibility File.

4.7.5.1.1 Pell Recipient Information - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for Pell Recipient Information interface is based on a bulk file transfer from COD to NSLDS system. The EAI development team provided parameters to Job Control Language (JCL) that initiates the file transfer. Data Integrator has been configured to manage the file transfers and imitate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.7.5.1.2 Pell Recipient Information – Test Scenario Dependencies

To execute the Pell Recipient Information interface test scenario the following dependencies were met:

- COD must have IBM MQSeries v2.1 and Commerce Quest Data Integrator v4.0.1 installed.
- COD must have MQSeries and Data Integrator running.
- COD must place components necessary to create the Pell Recipient Information file onto the appropriate MQSeries Queue.
- COD must send an MQSeries Message that will trigger the utility on the NSLDS system when the entire batch has been committed to MQSeries queues on the EAI Bus.
- EAI Bus must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- EAI Bus must have MQSeries and Data Integrator running.
- NSLDS must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- NSLDS must have MQSeries and Data Integrator running.
- NSLDS must accept the Pell Recipient Information file into an agreed upon directory for processing.
- NSLDS must make the agreed upon directory read/write-able for the Utility.
- NSLDS must provide a compiler for C++ for use by the EAI Team.

4.7.5.1.3 Pell Recipient Information - Test Scenario Inputs

The Pell Recipient Information interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries; it flows from COD to the EAI Bus to NSLDS. The EAI Bus is simply a pass through for the files and no transformation is performed. Two types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the COD, EAI Bus, and NSLDS.
2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the COD to NSLDS interface.

Refer to Appendix H for the detail test conditions.

4.7.5.1.4 Pell Recipient Information – Test Scenario Expected Results

The Release 3.0 EAI Core team executed this Pell Recipient Information interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix H for the expected results from the test conditions.

4.7.5.2 Pell Recipient Data Errors

The Pell Recipient Data Errors is transmitted from NSLDS to COD daily. The file reports all errors detected by NSLDS during the processing of the previous cycle's Pell Recipient Information file.

4.7.5.2.1 Pell Recipient Data Errors - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for Pell Recipient Data Errors interface is based on a bulk file transfer from NSLDS to COD system. The EAI development team provided parameters to Job Control Language (JCL) that initiates the file transfer. Data Integrator was configured to manage the file transfers and imitate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.7.5.2.2 Pell Recipient Data Errors – Test Scenario Dependencies

To execute the Pell Recipient Data Errors interface test scenario the following dependencies were met:

- NSLDS must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- NSLDS must have MQSeries and Data Integrator running.
- NSLDS must produce the Pell Recipient Data Errors file and place it into an agreed upon directory for processing.
- NSLDS must make the agreed upon directory read/write-able for the Utility.
- NSLDS must provide a compiler for C++ for use by the EAI Team.
- EAI Bus must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- EAI Bus must have MQSeries and Data Integrator running.
- COD must have IBM MQSeries v2.1 and Commerce Quest Data Integrator v4.0.1 installed.
- COD must have MQSeries and Data Integrator running.
- COD must have MQSeries running.
- COD must place components necessary to retrieve the Pell Recipient Data Errors from the appropriate MQSeries Queue.

4.7.5.2.3 Pell Recipient Data Errors - Test Scenario Inputs

The Pell Recipient Data Errors interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries; it flows from NSLDS to the EAI Bus to COD. The EAI Bus is simply a pass through for the files and no transformation is performed. Two types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the NSLDS, EAI Bus, and COD.
2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the NSLDS to COD interface.

Refer to Appendix H for the detail test conditions.

4.7.5.2.4 Pell Recipient Data Errors – Test Scenario Expected Results

The Release 3.0 EAI Core team executed this Pell Recipient Data Errors interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix H for the expected results from the test conditions.

4.7.6 COD to PEPS Overview

4.7.6.1 Daily Participants Feed

PEPS provides a daily feed of school data maintained within the PEPS system. This feed contains a full extract of all participant eligibility data contained in PEPS that is useful to COD processing. Although PEPS sends a complete extract on a daily basis (M-F, excluding holidays), COD is only interested in receiving a full update for initial data load and disaster recovery. At other times the feed is provided, COD will receive a daily batch containing only changes. This delta file will be created and sent by the PEPS MQSeries Utility, which runs on the PEPS Server, but outside of the PEPS Application.

Data Integrator uses a directory monitoring process to watch for the complete creation of a source flat file in a specified directory. The directory monitoring process looks at a specific directory for the creation or revision of any files that fit the naming pattern that it has been configured to look for.

4.7.6.1.1 Daily Participants Feed - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for Daily Participants Feed interface is based on a bulk file transfer from PEPS to COD system. The EAI development team uses Data Integrator’s directory monitor service to initiate the file transfer. Data Integrator has been configured to manage the file transfers and imitate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.7.6.1.2 Daily Participants Feed – Test Scenario Dependencies

To execute the Daily Participants Feed interface test scenario the following dependencies were met:

- PEPS must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- PEPS must have MQSeries, Data Integrator, and Directory Monitoring running.
- PEPS must place its Daily Participants Feed into an agreed upon source directory for processing.
- PEPS must make the agreed upon source directory readable for the Utility.
- PEPS must agree upon destination directory outside of the PEPS application space read/write-able for the Utility to create the Daily Participants Feed.
- PEPS must create a Java Runtime Environment (JRE v. 1.1.7) for the use of the PEPS MQSeries Utility outside of the PEPS Application.
- EAI Bus must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- EAI Bus must have MQSeries and Data Integrator running.
- COD must have IBM MQSeries v2.1 and Commerce Quest Data Integrator v4.0.1 installed.
- COD must have MQSeries and Data Integrator running.
- COD must accept the Daily Participants Feed file into an agreed upon directory for processing.
- COD must make the agreed upon directory read/write-able for the Utility.

4.7.6.1.3 Daily Participants Feed - Test Scenario Inputs

This Daily Participants Feed interface transports file via Data Integrator and MQSeries, it flows from PEPS to the EAI Bus, to COD. The EAI Bus is simply a pass through for the files and, no transformation is performed. Two types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the PEPS, EAI Bus, and COD.
2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the PEPS to COD interface.

Refer to Appendix I for the detail test conditions.

4.7.6.1.4 Daily Participants Feed – Test Scenario Expected Results

The Release 3.0 EAI Core team executed this Daily Participants Feed interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix I for the expected results from the test conditions.

4.7.7 COD to SAIG Overview

4.7.7.1 Common Record Schools Interface File

XML based Common Records are transported directly from the COD SAIG mailbox to COD. Common Record Acknowledgements and Responses from COD to Common Record schools are written to the SAIG mailbox of the school. Common Records do not require any transformation, but Common Record Acknowledgements destined for legacy record schools will need transformation in order for the schools to process the files.

4.7.7.1.1 Common Record Input - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for Common Record Input interface is based on a bulk file transfer from SAIG to COD system. The EAI development team provided parameters for COD to poll SAIG mailbox that initiates the file transfer. Data Integrator has been configured to manage the file transfers and imitate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.7.7.1.2 Common Record Input – Test Scenario Dependencies

To execute the Common Record Input interface test scenario the following dependencies were met:

- SAIG must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- SAIG must have MQSeries and Data Integrator running.
- SAIG must produce the Common Record Input file and place it into an agreed upon directory for processing.
- SAIG must make the agreed upon directory read/write-able for the Utility.
- EAI Bus must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- EAI Bus must have MQSeries and Data Integrator running.
- COD must have IBM MQSeries v2.1 and Commerce Quest Data Integrator v4.0.1 installed.
- COD must have MQSeries and Data Integrator running.
- COD must have MQSeries running.
- COD must place components necessary to retrieve the Common Record Input from the appropriate MQSeries Queue.

4.7.7.1.3 Common Record Input - Test Scenario Inputs

The Common Record Input interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries; it flows from SAIG to the EAI Bus to COD. The EAI Bus is simply a pass through for the files and no transformation is performed. Two types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the SAIG, EAI Bus, and COD.
2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the SAIG to COD interface.

Refer to Appendix J for the detail test conditions.

4.7.7.1.4 Common Record Input – Test Scenario Expected Results

The Release 3.0 EAI Core team executed this Common Record Input interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix J for the expected results from the test conditions.

4.7.7.1.5 Common Record Acknowledgements and Responses - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for Common Record Acknowledgements and Responses interface is based on a bulk file transfer from COD to SAIG system. The EAI development team provided parameters to Job Control Language (JCL) that initiates the file transfer. Data Integrator has been configured to manage the file transfers and imitate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.7.7.1.6 Common Record Acknowledgements and Responses – Test Scenario Dependencies

To execute the Common Record Acknowledgements and Responses interface test scenario the following dependencies were met:

- COD must have IBM MQSeries v2.1 and Commerce Quest Data Integrator v4.0.1 installed.
- COD must have MQSeries and Data Integrator running.
- COD must place its Common Record Acknowledgements and Responses into an agreed upon source directory for processing.
- COD must make the agreed upon source directory readable for the Utility.
- EAI Bus must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- EAI Bus must have MQSeries and Data Integrator running.
- EAI Bus must have Java Runtime Environment and Java transformation code.
- SAIG must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- SAIG must have MQSeries and Data Integrator running.
- SAIG must accept Common Record files into an agreed upon directory for processing.
- SAIG must make the agreed upon directory read/write-able for the Utility.

4.7.7.1.7 Common Record Acknowledgements and Responses - Test Scenario Inputs

The Common Record Input Acknowledgements and Responses interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries; it flows from COD to the EAI Bus to SAIG. The EAI Bus transforms the files and passes the files. Two types of test data were used in the system integrated test phase:

1. Common Record Acknowledgement files from COD were transformed and sent to Common Record schools' mailbox on SAIG.
2. Common Record Response files from COD were transformed and sent to Common Record schools' mailbox on SAIG.

Refer to Appendix J for the detail test conditions.

4.7.7.1.8 Common Record Acknowledgements and Responses – Test Scenario Expected Results

The Release 3.0 EAI Core team executed this Common Record Acknowledgements and Responses interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix J for the expected results from the test conditions.

4.7.7.2 Legacy Record Schools Interface File

Schools produce the Pell Origination and Disbursement Records in Legacy format. Types of records transported from COD to SAIG include Pell Acknowledgement Records and Pell Disbursement Acknowledgement Records. Schools produce the Direct Loan Origination Record, Direct Loan Origination Change, PLUS Origination, and Direct Loan Disbursement Records. Responses from COD were written to the SAIG mailbox of the school. Types of records transported from COD to SAIG include Direct Loan Origination Acknowledgement Record, Direct Loan Origination Change Acknowledgement Record, PLUS Origination Acknowledgement Record, MPN/PLUS P-Note Acknowledgement, PLUS Credit Decision, Booking Notification, Common Records, Promissory Note Acknowledgement, and Direct Loan Disbursement Acknowledgement Record.

When COD receives a Legacy Record input file directly from SAIG, COD generates a XML based Common Record acknowledgement for each Legacy Record received. When COD processes the input file Legacy Record and validates its contents, COD will create a XML based Common Record response. The EAI BUS will need to transform Common Record Acknowledgements and Responses to Legacy format for Legacy Record schools before placing them in the appropriate SAIG mailbox.

4.7.7.2.1 Legacy Record Input - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for Legacy Record Input interface is based on a bulk file transfer from SAIG to COD system. The EAI development team provided parameters for COD to poll SAIG mailbox that initiates the file transfer. Data Integrator has been configured to manage the file transfers and imitate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.7.7.2.2 Legacy Record Input – Test Scenario Dependencies

To execute the Legacy Record Input interface test scenario the following dependencies were met:

- SAIG must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- SAIG must have MQSeries and Data Integrator running.
- SAIG must produce the Legacy Record Input file and place it into an agreed upon directory for processing.
- SAIG must make the agreed upon directory read/write-able for the Utility.
- EAI Bus must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- EAI Bus must have MQSeries and Data Integrator running.
- COD must have IBM MQSeries v2.1 and Commerce Quest Data Integrator v4.0.1 installed.
- COD must have MQSeries and Data Integrator running.
- COD must have MQSeries running.
- COD must place components necessary to retrieve the Legacy Record Input from the appropriate MQSeries Queue.

4.7.7.2.3 Legacy Record Input - Test Scenario Inputs

The Legacy Record Input interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries; it flows from SAIG to the EAI Bus to COD. The EAI Bus is simply a pass through for the files and no transformation is performed. Two types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the SAIG, EAI Bus, and COD.
2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the SAIG to COD interface.

Refer to Appendix J for the detail test conditions.

4.7.7.2.4 Legacy Record Input – Test Scenario Expected Results

The Release 3.0 EAI Core team executed this Legacy Record Input interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix J for the expected results from the test conditions.

4.7.7.2.5 Legacy Record Acknowledgements and Responses - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for Legacy Record Acknowledgements and Responses interface is based on a bulk file transfer from COD to SAIG system. The EAI development team provided parameters to Job Control Language (JCL) that initiates the file transfer. Data Integrator has been configured to manage the file transfers and imitate the start of the application process at different stages (e.g., EAI Bus) of the file transfer.

Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.7.7.2.6 Legacy Record Acknowledgements and Responses – Test Scenario Dependencies

To execute the Legacy Record Input interface test scenario the following dependencies were met:

- COD must have IBM MQSeries v2.1 and Commerce Quest Data Integrator v4.0.1 installed.
- COD must have MQSeries and Data Integrator running.
- COD must place its Common Record Acknowledgements and Responses into an agreed upon source directory for processing.
- COD must make the agreed upon source directory readable for the Utility.
- EAI Bus must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- EAI Bus must have MQSeries and Data Integrator running.
- EAI Bus must have Java Runtime Environment and Java transformation code.
- SAIG must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- SAIG must have MQSeries and Data Integrator running.
- SAIG must accept Legacy Record files into an agreed upon directory for processing.
- SAIG must make the agreed upon directory read/write-able for the Utility.

4.7.7.2.7 Legacy Record Acknowledgements and Responses - Test Scenario Inputs

The Legacy Record Acknowledgements and Responses interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries; it flows from COD to the EAI Bus to SAIG. The EAI Bus transforms the files and passes the files. Two types of test data were used in the system integrated test phase:

1. Common Record Acknowledgement files from COD were transformed and sent to Legacy Record schools' mailbox on SAIG.
2. Common Record Response files from COD were transformed and sent to Legacy Record schools' mailbox on SAIG.

Refer to Appendix J for the detail test conditions.

4.7.7.2.8 Legacy Record Acknowledgements and Responses – Test Scenario Expected Results

The Release 3.0 EAI Core team executed this Legacy Record Acknowledgements and Responses interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix J for the expected results from the test conditions.

4.7.7.2.9 School Destination Information Feed

The Participant Management System provides destination and mailbox data for COD. This will replace the Institution, Destination, and Participation File currently sent from TIVWAN.

4.7.7.2.10 School Destination Information Feed - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for School Destination Information Feed interface is based on a bulk file transfer from SAIG to COD system. The EAI development team provided parameters for COD to poll SAIG mailbox that initiates the file transfer. Data Integrator has been configured to manage the file transfers and imitate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.7.7.2.11 School Destination Information Feed – Test Scenario Dependencies

To execute the School Destination Information Feed interface test scenario the following dependencies were met:

- SAIG must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- SAIG must have MQSeries and Data Integrator running.
- SAIG must produce the School Destination Information Feed file and place it into an agreed upon directory for processing.
- SAIG must make the agreed upon directory read/write-able for the Utility.
- EAI Bus must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- EAI Bus must have MQSeries and Data Integrator running.
- COD must have IBM MQSeries v2.1 and Commerce Quest Data Integrator v4.0.1 installed.
- COD must have MQSeries and Data Integrator running.
- COD must have MQSeries running.
- COD must place components necessary to retrieve the School Destination Information Feed from the appropriate MQSeries Queue.

4.7.7.2.12 School Destination Information Feed - Test Scenario Inputs

The School Destination Information Feed interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries; it flows from SAIG to the EAI Bus to COD. The EAI Bus is simply a pass through for the files and no transformation is performed. Two types of test data were used in the system integrated test phase:

1. A test file was created by the EAI team to test the connectivity between the SAIG, EAI Bus, and COD.

2. Test files (with estimated production size data) were used by the application and EAI team to test the volume and the load through the SAIG to COD interface.

Refer to Appendix J for the detail test conditions.

4.7.7.2.13 School Destination Information Feed – Test Scenario Expected Results

The Release 3.0 EAI Core team executed this School Destination Information Feed interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix J for the expected results from the test conditions.

4.7.7.3 COD to Schools/SAIG Reports

COD produces reports when requested by the schools via the web interface. These reports will be delivered to the schools' SAIG mailbox. The reports include Electronic Summary of Account (ESOA), Multiple Reporting Records, Year-to-date Disbursements, Loan Servicer Refunds, Rebuild Data File, Text Messages, Reconciliation, Certification Warning Requirement, Anticipated Disbursement Listing, Actual Disbursement Roster, Inactive Loans Report, SSN/Name/DOB Change Report, Loan Data Matching Exception Report, Duplicate Student Borrower Report, Disbursement activity Not Yet Booked at Servicing, Direct Loan COD Combo Report.

4.7.7.3.1 COD to Schools/SAIG Reports - Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure for COD to Schools/SAIG Reports interface is based on a bulk file transfer from COD to SAIG system. The EAI development team provided parameters to Job Control Language (JCL) that initiates the file transfer. Data Integrator has been configured to manage the file transfers and imitate the start of the application process at different stages (e.g., EAI Bus) of the file transfer. Data Integrator leverages MQSeries to send large files as MQ messages and load balance the messages over multiple MQSeries queues.

4.7.7.3.2 COD to Schools/SAIG Reports – Test Scenario Dependencies

To execute the COD to Schools/SAIG Reports interface test scenario the following dependencies were met:

- COD must have IBM MQSeries v2.1 and Commerce Quest Data Integrator v4.0.1 installed.
- COD must have MQSeries and Data Integrator running.
- COD must place its COD to Schools/SAIG Reports into an agreed upon source directory for processing.
- COD must make the agreed upon source directory readable for the Utility.
- EAI Bus must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.
- EAI Bus must have MQSeries and Data Integrator running.
- EAI Bus must have Java Runtime Environment and Java transformation code.
- SAIG must have IBM MQSeries v5.2 and Commerce Quest Data Integrator v4.0.1 installed.

- SAIG must have MQSeries and Data Integrator running.
- SAIG must accept COD to Schools/SAIG Reports files into an agreed upon directory for processing.
- SAIG must make the agreed upon directory read/write-able for the Utility.

4.7.7.3.3 COD to Schools/SAIG Reports - Test Scenario Inputs

The COD to Schools/SAIG Reports interface test scenario includes three test cycles: 1) Normal 2) Expected Error 3) Unexpected Error. This interface transports files via Data Integrator and MQSeries; it flows from COD to the EAI Bus to SAIG. The EAI Bus transforms the files and passes the files. One type of test data was used in the system integrated test phase:

1. COD to Schools/SAIG Reports files were transformed and sent to schools' mailbox on SAIG.

Refer to Appendix J for the detail test conditions.

4.7.7.3.4 COD to Schools/SAIG Reports – Test Scenario Expected Results

The Release 3.0 EAI Core team executed this COD to Schools/SAIG Reports interface test scenario and the expected results were received and validated. All actual results matched the expected results. Refer to Appendix J for the expected results from the test conditions.

5 SOURCE CODE FOR NEW ADAPTERS

Due to the large volume of source code, EAI is not including source code in this deliverable. Source code does exist and is available upon request. Please contact Patrick.E.Volpe@accenture.com for source code.

6 APPENDICES

Appendix A - A Road Map between Work and Deliverables

Appendix B – FPDM Assembly Test Conditions.xls

Appendix C – eCB Assembly Test Conditions.xls

Appendix D – CPS Assembly Test Conditions.xls

Appendix E – DLOS Assembly Test Conditions.xls

Appendix F – DLSS Assembly Test Conditions.xls

Appendix G – FMS Assembly Test Conditions.xls

Appendix H – NSLDS Assembly Test Conditions.xls

Appendix I – PEPS Assembly Test Conditions.xls

Appendix J – SAIG Assembly Test Conditions.xls

Appendix K – Common Logging Test Conditions.xls

Appendix L – Timeline.ppt